15 PDS 软件中 DDR3 存储器 IP 配置和使用

工程源码	
相关视频课程	

章节导读

本节将主要介绍 DDR3 的基本工作原理和用途，同时介绍 PDS 软件的 DDR 控制器 IP 的创建流程、IP 用户使用接口 AXI 协议介绍和 IP 对应的 Example Design 的仿真和上板验证流程。

15.1RAM 存储器发展革沿

DDR 存储器发展到今天，无非是在两个方向上不断探索前进。一是追求更高的存储容量，二是追求更快的读写速度。虽然 DDR3 已不是当今主流的 DDR 存储器，市场上的 DDR4 和 DDR5 也已经层出不穷。但是 DDR3 存储器作为 RAM 存储器家族发展历程中的一个重要里程碑，其学习价值也是非常丰厚的。通过对本小节内容的学习，读者既可以把握 RAM 存储器的发展路线，又可以明确当代主流 DDR 存储器的使用场合，同时，这些知识和内容，还可以作为读者了解更高等级 DDR 存储器的一块敲门砖。

话说 DDR 存储器的发展革沿，得从追溯到早期的 SRAM 存储器开始。我们在数电的课程中也了解过，使用 6 个晶体管的组合，可以实现 1 位数据的存储。

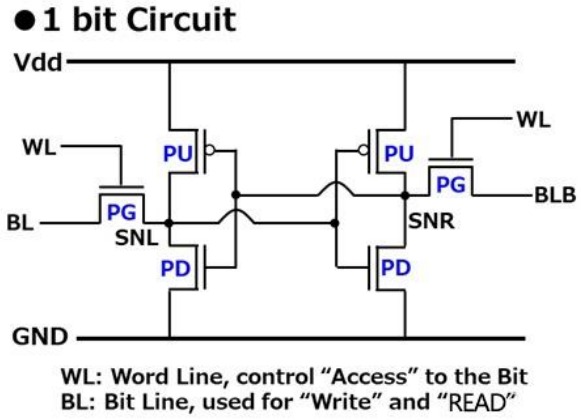


图 15-1 SRAM 使用 6 个晶体管实现 1 位数据的存储示意图

因此，早期 SRAM 芯片对于晶体管的消耗量是巨大的。而影响芯片的成本最大的一个因素就是芯片内部集成的晶体管数量。正是受到 RAM 存储器单位存储容量过高成本的制约，工程师们便想到通过优化芯片内部存储器件的方法，

来降低芯片制造成本，只有通过降低单位 RAM 存储器的制造成本，才能让市场接受更大存储容量的存储器价格。而工程师们最终寻找到的突破口，就是使用 1 个电容+1 个晶体管的组合，实现 1 位数据的存储。

如果给电容两端施加电压，电容两端就会形成一个电势差。如果电容里面储存有电荷，那么这两端就会有电压差，此时，我们就可以认为值存储为 1；如果电荷电容器里面没有电压差，则说明该电容内没有储存电荷，那么这个时候我们就认为它存储的数据为 0。这样，原来的 SRAM 从依靠 6 个晶体管存储 1 位数据，到现在仅需要 1 个晶体管配合 1 个电容就可以存储 1 位数据，单位 RAM 存储容量的制造成本就大幅下降了。而 1 个晶体管配合 1 个电容就是 SDRAM 的 1bit 数据存储硬件开销。现如今，SDRAM 仍然在当前部分缓存容量和缓存数据要求不高的场合，凭借其成本优势具有一定实用价值。

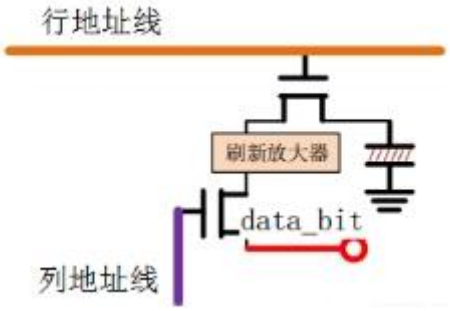


图 15-2 SDRAM 存储 1bit 数据示意图

正是存储结构上的创新，存储数据原理的变化，解决了 SRAM 无法做到大容量经济性这样一个问题。因此 SDRAM 最大的特点就是容量大。虽然 SDRAM 相对于 SRAM 的存储容量有大幅提升，但这个提升也仅仅是相对的，从发展的眼光来看，还是无法满足当今日新月异的电子设备数据缓存需求。况且，虽然 SDRAM 相比于 SRAM 在容量得到了提升，但是这一轮进化在器件工作频率的提升效果上，并不明显。这样，RAM 存储器的发展，就来到了 DDR SDRAM 设计阶段。

相较于 SDRAM，DDR SDRAM 的核心技术在于存储单元的读写速率和存储器接口的读写速率分离。

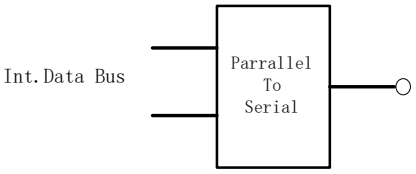
在 SDRAM 阶段，存储单元的读写速率和存储接口的读写速率近乎相同，且存储单元的读写速率也遇到读写速率的瓶颈，很难再有提升空间。下一步，只能通过在存储单元的存储周期不变的情况下，通过改变整个存储器的接口设计策略，让存储器接口在同一个存储单元的存储周期读写 2bit 的数据，同时，重新设计存储器接口，让存储器接口的读写速率为存储单元的读写速率的 2 倍。

这就是 DDR SDRAM 的核心设计思想。

从时钟的角度来分析，DDR SDRAM 在发挥时钟信号的工作效率上作文章，虽然存储单元的读写速率很难提升，但经过优化后的存储器接口，可以让数据的交互既发生在时钟的上升沿，又发生在时钟的下降沿，同时，将存储单元内的 2bit 数据作为操作对象。通过这样一种改进，一个存储器它的吞吐速率就变成了原来的两倍。

DDR2 理论上来说，它和第一代 DDR SDRAM 在结构原理上区别不大，只是对存储器的接口速率进行了一些优化。DDR2 接口以 200M（400M 双沿等效）时钟传输数据，同时采用了 4bit 预读取技术，实现了接口速率优化目标。

DDR3 相比于 DDR2 来说，更加改进了存储器接口的时钟可耐受频率，同时，做到了更低的功耗。DDR3 接口以 400M（800M 双沿等效）时钟传输数据，同时采用了 8bit 预读取技术。由于每个存储器读写周期（200M）可以包含 4 个双沿等效后的接口时钟周期（800M），这样每个存储器读写周期可以读/写 32bit



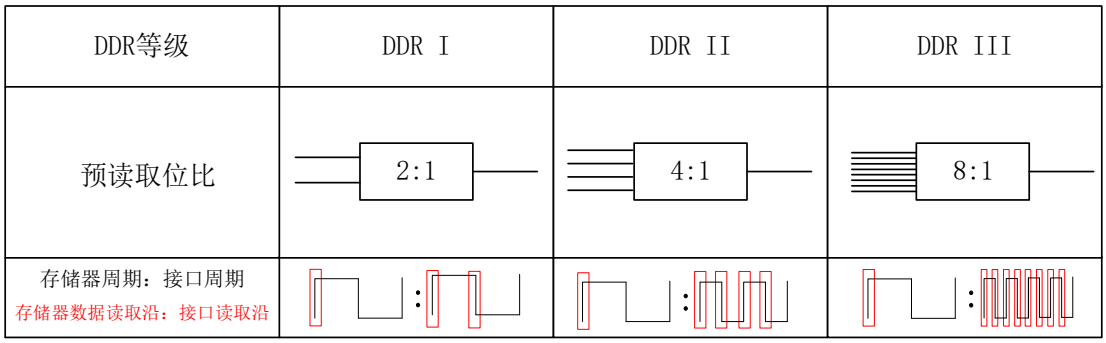


图 15-3 DDR 存储器迭代与数据预读取量、接口读取周期对应关系

这里，有读者就有疑问了。如果不提高 DDR 存储器的接口制造工艺，而单纯的给以更高的接口读写时钟频率，这样是否可行呢？回答是否定的。当存储器件的接口工艺没有换代升级，存储器接收到更高频率的读写时钟信号后，很有可能会出现时序的紊乱，即数据存储时序无法收敛。

这样，介绍完 RAM 存储器的历史革沿，不断发展的 DDR 存储器通过迭代升级，既实现了存储容量的增长，又实现了存储速率的增长。

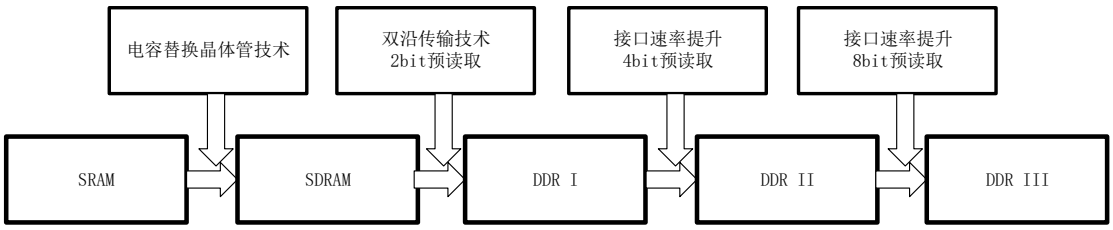


图 15-4 RAM 存储器发展革沿概略图

讲到这里，我们可以一起来举例分析一下，在综合成本、存储容量和读写速率的各因素条件下，如何选择缓存器件呢？

举例来说，我们想进行一次 1080P 的 30FPS 图像数据 5 秒钟的存储。那么需要多大的存储容量呢？经过计算，一帧 1080P 图像需要 5MByte 的存储空间，那么完成上述任务，就需要 150MByte 的存储空间。而这样一个存储空间的选择，从存储容量，存储速率和经济成本各个角度来权衡，使用 DDR3 存储器是合理可行的。

在我们后续的课程讲解中，基于 FPGA 的 DDR3 应用案例，主要有两大类。第一类是图像传输缓存及图像处理系统，第二类是高速数据采集系统的应用。

第一类应用案例，主要是利用 DDR3 的大容量存储空间用于存放临时数据，在工程设计中，这些临时数据会被用来二次加工和提取关键信息，而加工和提取的过程，又有可能涉及到存储容量的消耗。第二类应用案例，主要是解决数据采集与发送的速率差异。由于数据收发速率的不同，如果不加控制处理，数据的读出必然会存在溢出（写的快）或间歇（读的快）的现象，DDR3 存储器通过将采集到的数据缓存起来，数据能做到集中采集，集中发送，这样，数据采集的可靠性就会获得大大提高。数据采集之所以要用到 DDR3，第一因为它数据存储量大，第二它的数据读写速率快。

随着技术的迭代升级，DDR 存储器的控制信号越来越多，管脚控制也越来越复杂。然而作为用户，我们并不需要自己去设计 DDR 存储器的控制器。Xilinx 官方已经给我们提供了成熟的 DDR3 控制器 IP 核，我们可以通过创建 DDR3 控制器 IP 核，实现对 DDR3 控制器的存储控制。用户在实际应用中，只需要关心对存储器写入和读出的数据，关心一些基本的时钟、复位和使能信号，即可实现对数据存储的操作。

15.2DDR3 控制器 IP 创建流程

新建 PDS 工程，首先在菜单栏里选择“Tools”然后单击 “IP Compiler”选

项，如图 15-5 所示。

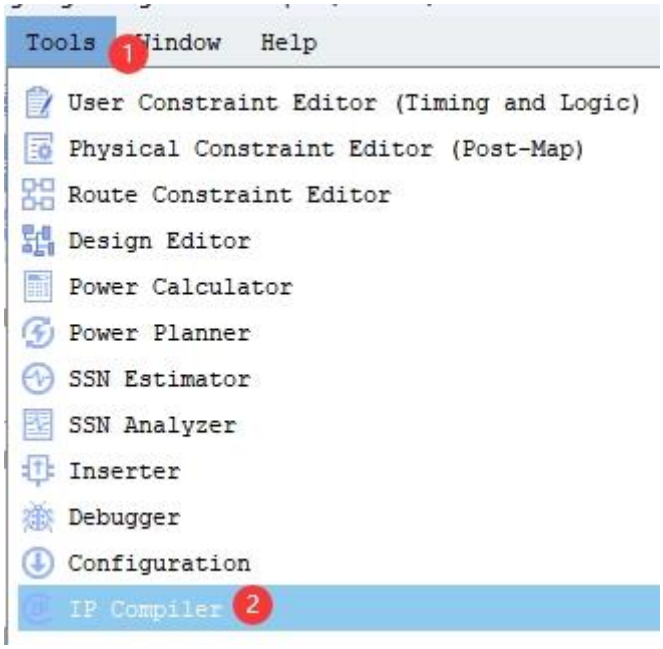


图 15-5“IP Compiler”页面

点击图 15-5 中的“IP Compiler”后会跳转图 15-6 页面。由于 PDS 软件安装时默认是不带有 DDR3 IP 核，所以我们需要将 DDR3 IP 手动加载进来。

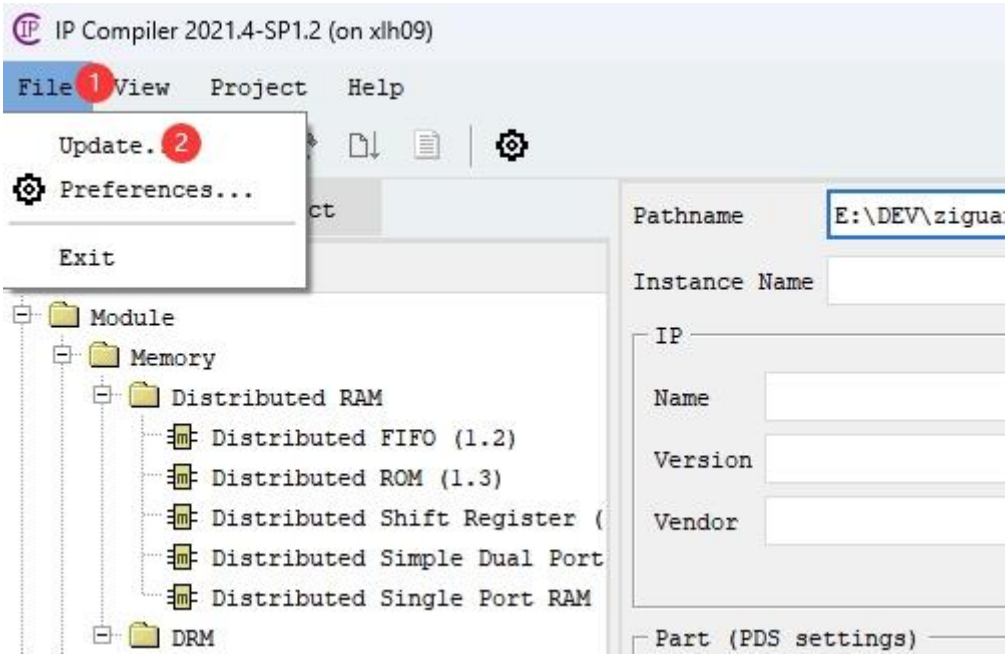


图 15-6 Update IP 页面

加载 DDR3 IP 方法如下：点击图 15-6 中标签 1 中的“File” 按钮；然后再点击标签 2 中的“Update...”按钮，点击图 15-6 页面中的“Update...”按钮之后会自动弹出图 15-7 页面。

图 15-7 加载 DDR3 IP

点击图 15-7 页面中箭头指向的“Add Packages”按钮，进入图 15-8 页面中。本章实验所需的代码放在对应工程文件夹下面的 ipcore 文件夹下面，选中图 15-8 中箭头指向的 1_ipsl_hmic_h_v1_1.iar 文件，然后点击图 15-8 箭头指向的“open”按钮。

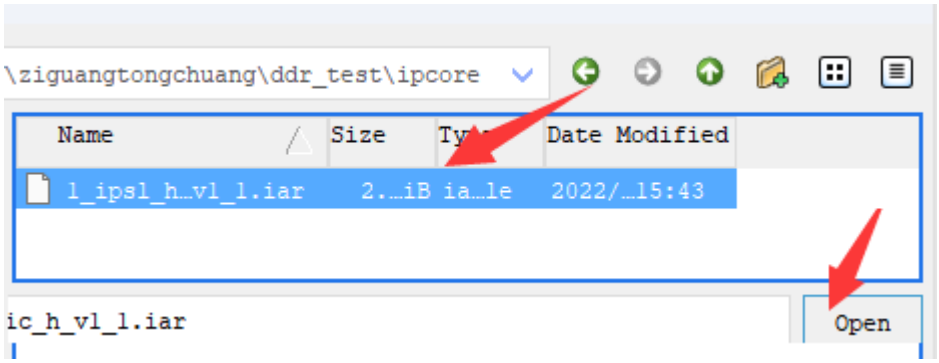


图 15-8 1_ipsl_hmic_h_v1_1.iar 所在路径

点击图 15-8 中的“open”按钮之后会自动跳转到图 15-9 页面中。点击图 15-9 中箭头指向的方框，方框会自动加载一个对号，然后点击图 15-9 中的“Install”按钮。

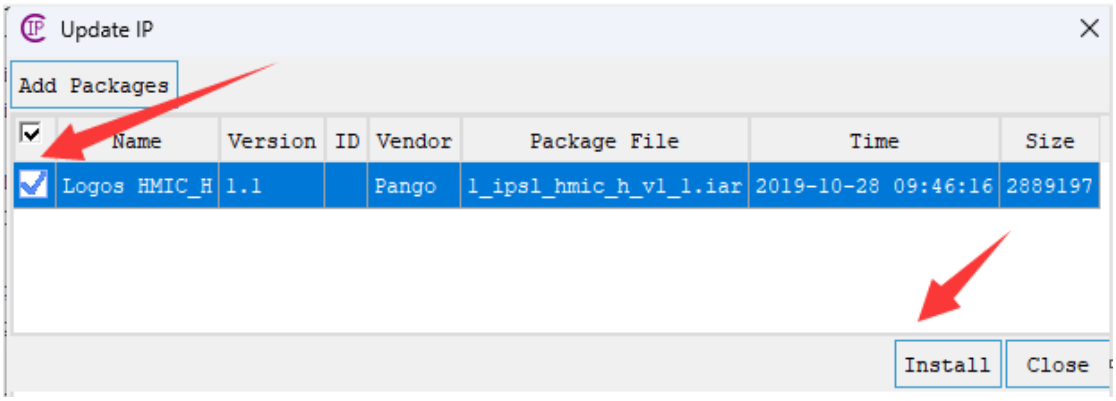


图 15-9 加载 DDR3 IP 核

点击图 15-9 中的“Install”按钮之后会自动返回到 IP 选择页面中，在该页面标签 1 中可以看到已经添加了新的 IP“Logos HMIC_H(1.1)”，点击选择 Lo

gos HMIC_H(1.1)，然后将标签 2 中“Instance Name”选项命名为“DDR3”。最后点击标签 3 中的“Customize”按钮进入 DDR3 IP 参数配置页面。

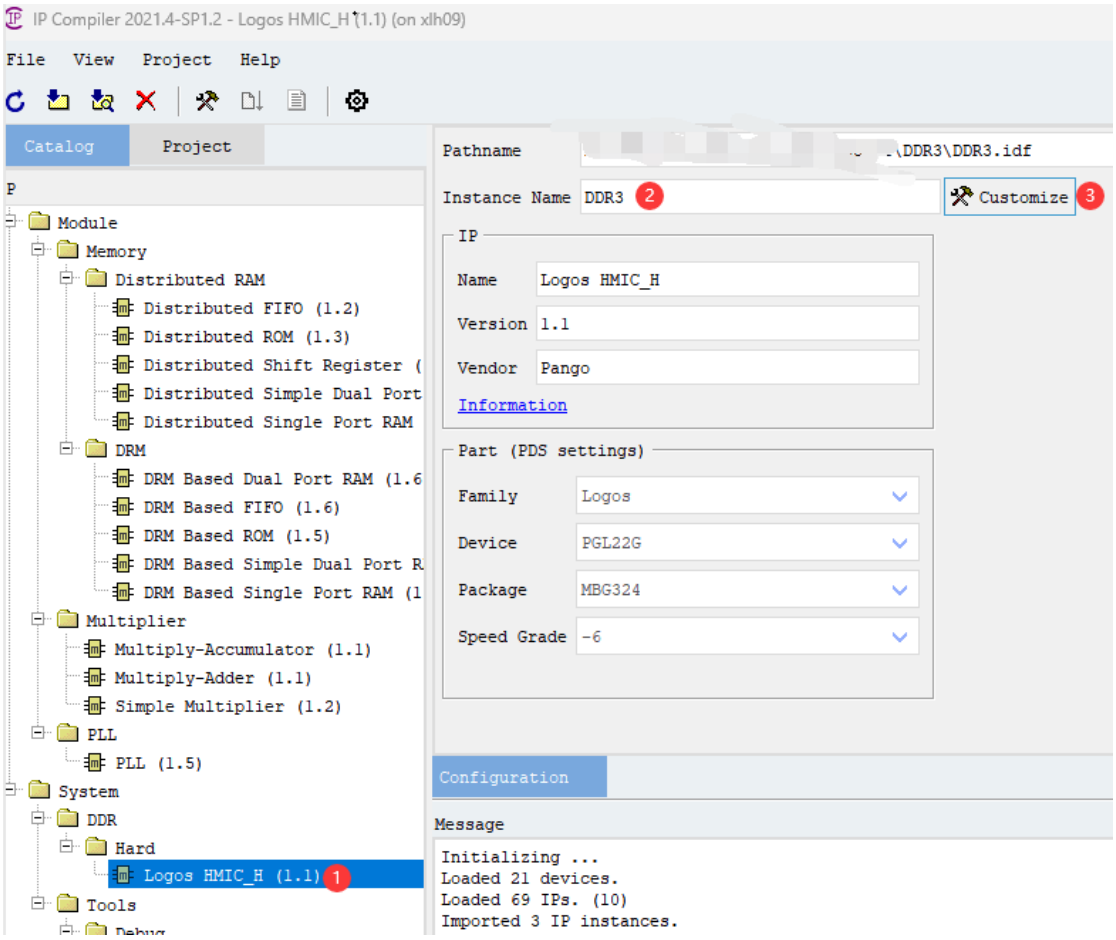


图 15-10 IP 选择页面

点击图 15-10 中的“Customize”按钮进入图 15-11 中。

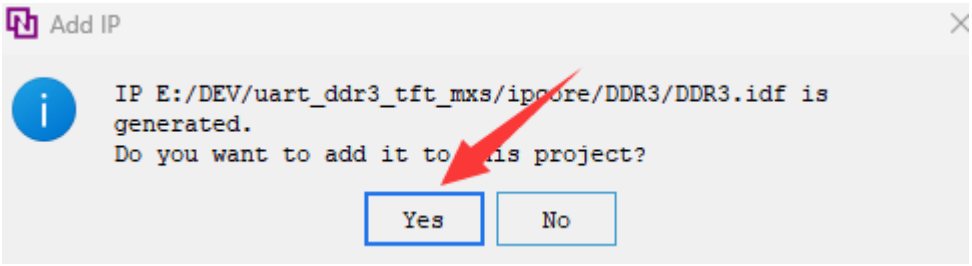


图 15-11“Add IP”弹窗

图 15-11 中表示的是 DDR3.idf 文件已经生成，询问我们是否将该文件加载到我们的工程中。因为这个 IP 我们后续是要在工程中用到的，因此这里直接点击图 15-11 中的“Yes”按钮即可。然后软件会自动 DDR3 的参数配置界面，如图 15-12 所示。

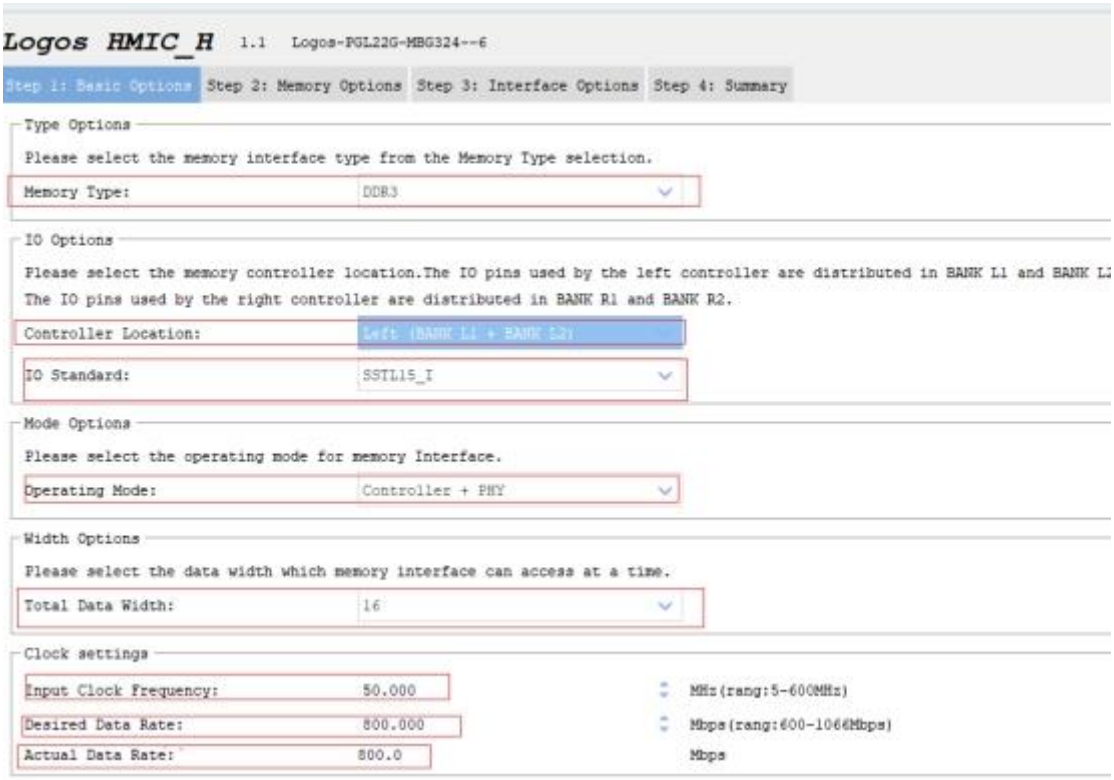


图 15-12“ Basic Options ”选项卡设置

Memory Type：目前支持的类型有 DDR3、DDR2 和 LPDDR。这里我们选择 DDR3 。

Controller Location：Controller 在 FPGA 芯片的位置，目前 PGL22 支持的位置有 Right (BANK R1 + BANK R2)和 Left (BANK L1 + BANK L2)两种, 因为开发板的硬件设计中，DDR3 芯片接到的 BANK L1 和 BANK L2，所以这里我们选择 Lift (BANK L1 + BANK L2)。

IO Standard：接口标准选项，DDR3 支持的接口标准有 SSTL15_I 和 SSTL 15_II 两种，这里我们保持默 认即可。

Operating Mode：HMIC_H 运行模式选择。目前只支持 Controller + PHY 模式，所以保持默认即可。

Total Data Width：与 HMIC_H 连接的片外 SDRAM 总共的 DQ 宽度，目前支持的总宽度有 8 和 16。由于本章我们的数据位宽为 16，所以这里保持默认值 16 即可。

Input Clock Frequency：HMIC_H 的输入时钟，单位 MHz。和开发板板载晶振频率保持一致，因为板载晶振频率为 50MHz，默认值也为 50MHz，所以此处保持默认即可。

Desired Data Rate：期望的数据速率， DDR3 支持的最高速率为 1066Mbps。保持默认即可，因为 DDR3 是双沿触发，即在时钟的上升沿和下降沿都能进行数据采集和发送，所以数据时钟配置实际上为数据速率的一半，例如：800Mbps 的数据速率，实际上只需要输出 400MHz 的时钟即可。

Actual Data Rate：实际能达到的数据速率，尽可能接近期望的速率。保持默认即可。

将图 15-12 中的参数设置完毕之后，我们再点击图 15-12 页面中的“Step 2: Memory Options”按钮，进入图 15-13“Step 2: Memory Options 选项卡设置”。

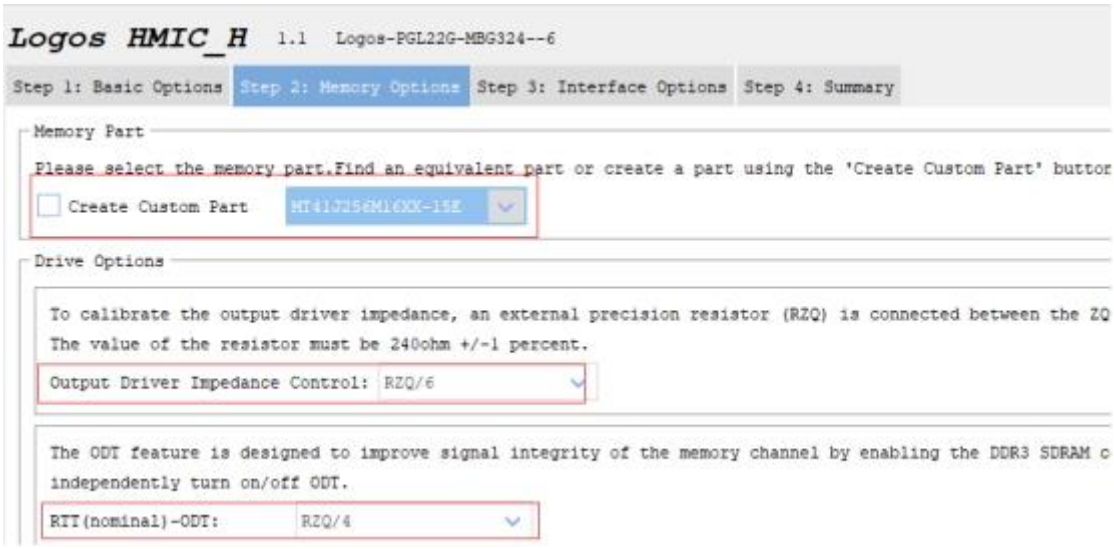


图 15-13“Step 2: Memory Options”选项卡设置

Memory Part：SDRAM 芯片的具体型号，所以这里我们选择 MT41J256M16XX-15E。

Drive Options：驱动能力选项，DDR3 支持的驱动选项为 Output Driver Impedance Control 和 T(nominal)-ODT。是内部高性能 bank 端接匹配阻抗的设置，这里不去改它，选择默认即可。 将图 15-13 中的参数设置完毕之后，我们再点击图 15-13 页面中的“Step 3: Interface Options”按钮，进入图 15-14“Step 3: Interface Options”选项卡设置。

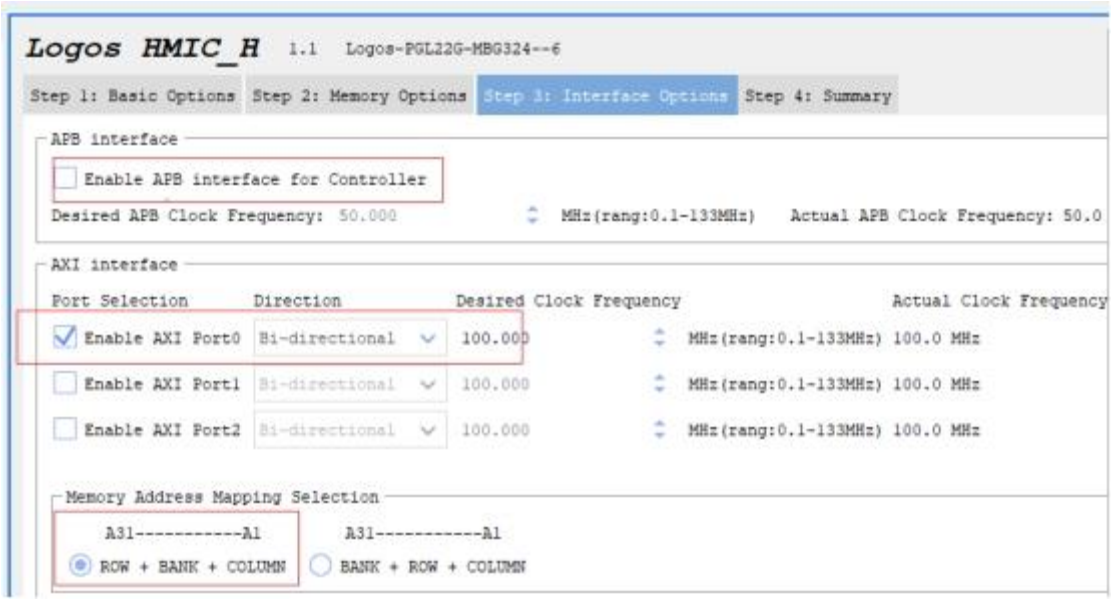


图 15-14“Step 3: Interface Options”选项卡设置

APB interface：APB 接口配置，包括使能和 APB 接口的时钟频率。APB(Advanced Peripheral Bus)外围 总线，是一种常见的总线协议。APB 属于 AMBA 3 协议系列，它提供了一个低功耗的接口，并降低了接口 的复杂性。APB 接口用在低带宽和不需要高性能总线的外围设备上。APB 是非流水线结构，所有的信号仅与时钟上升沿相关，这样就可以简化 APB 外围设备的设计流程，每个传输至少耗用两个周期。APB 可以与 AMBA 高级高性能总线(AHB-Lite)和 AMBA 高级可扩展接口(AXI)连接。因为这里我们没有用到 APB 接口，所以对于“Enable APB interface Controller”选项不用勾选。

AXI interface: 三组 AXI4 接口的配置，包括使能、读写方向、时钟频率。这里我们只用到了第一组接口，所以保持默认启用 AXI Port0 即可。 读写方向配置有三种可以选择的方式： 1） Bi-directional 可读可写 ； 2） Read 只读 ； 3） Write 只写 ； 读写方向这里我们保持“ Bi-directional”默认设置即可。

Memory Address Mapping Selection: SDRAM 的地址与 AXI4 地址映射选项。这里有两种选择方式:1） ROW + BANK + COLUMN ； 2） BANK + ROW + COLUMN

AXI4 的读写地址为 32 位，但是只有 A1-A31 有效，A0 为无效位。例如，选择 ROW+BANK+COLUMN 时，SDRAM 列地址 c0 对应 AXI 地址 A1，c1 对

应 A2，其它地址以此类推。　这里我们保持默认设置 ROW+BANK+COLUMN 即可 将图 15-14 中的参数设置完毕之后，我们再点击图 15-14 页面中的“Step 4: Summary”按钮，进入图 15-15“Step 4: Summary”页面。该页面用于打印当前的配置信息，不需要配置参数，页面如图 15-15 所示。

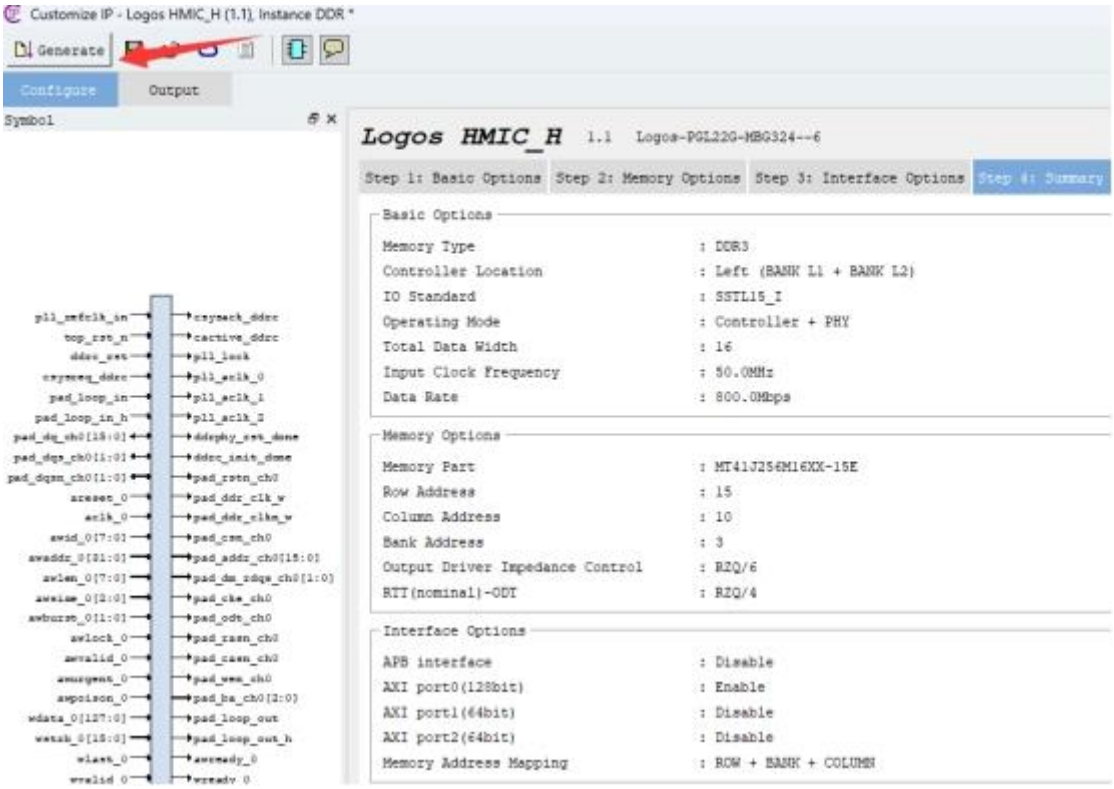


图 15-15“Step 4: Summary”页面

在图 15-15 中我们可以核对 DDR3 IP 核配置信息，配置完成之后再点击图 15-15 中的“Generate”按钮。之后会弹出图 15-16 页面，图 15-16 中表示的是：对 IP 核设置的数据将会保存，是否继续操作。我们继续点击图 15-16 中箭头指向的“OK”按钮。

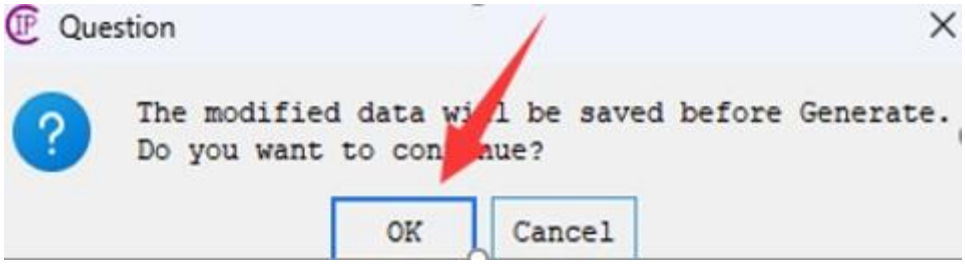


图 15-16“Question”页面

最终会弹出图 15-17，至此 DDR3 IP 核配置成功。

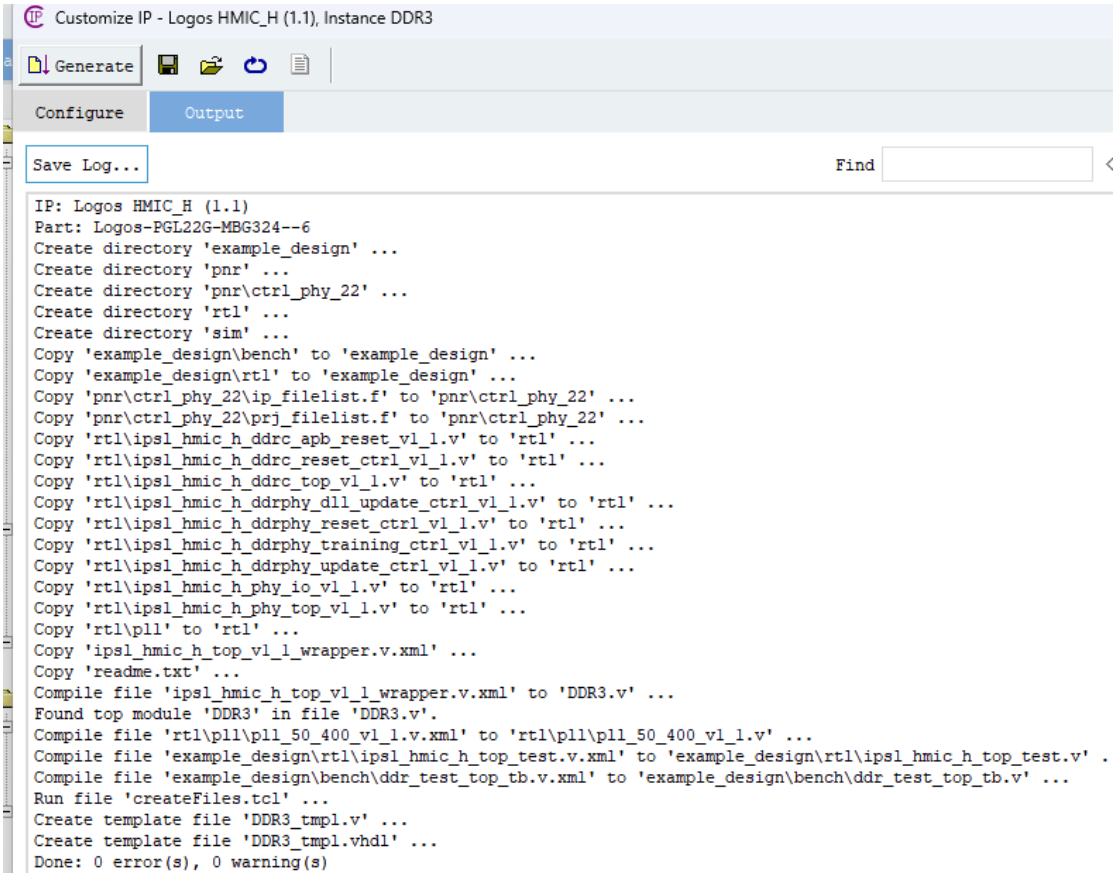


图 15-17 DDR3 IP 配置成功

IP 核配置成功后会自动弹出图 15-18 即这个 IP 的例化模板文件，接下来我们需要例化这些自动生成的代码（由于代码太多，所以在图 15-18 中并没有将自动生成的代码全部展示）。

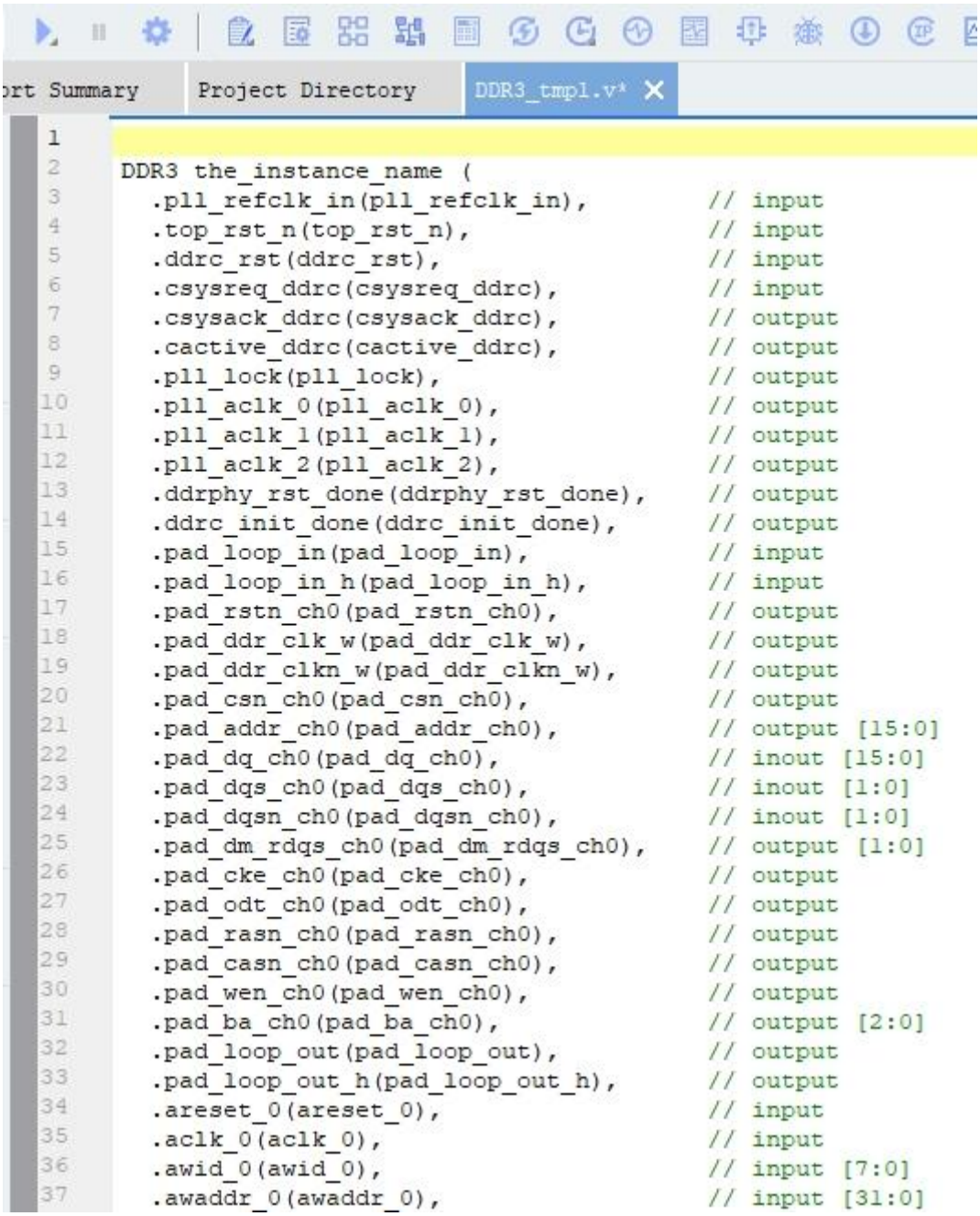


图 15-18 例化模板文件

将 IP 核配置过程中弹出的界面关闭，返回 Source 面板，如图 15-19 所示。

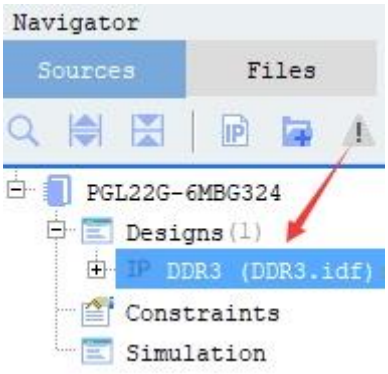


图 15-19 Source 面板

点击图 15-19 页面中 DDR3 IP 左侧加号位置，可以看到 DDR3 IP 核下有多个文件，点击图 15-20 中箭头所指的位置，打开 DDR3.v 文件。

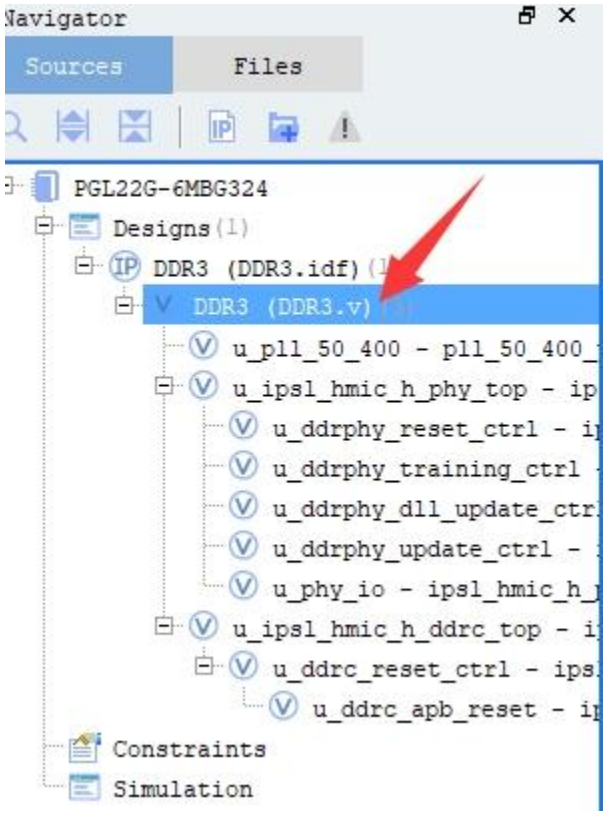


图 15-20 IP 核包含的文件

部分 DDR3.v 文件内容如图 15-21 所示（由于代码太多，所以并没有将自动生成的代码全部展示），若要调用 DDR3 IP，我们可以选择将图 15-21 中的端口例化到需要调用 DDR3 IP 的模块中，或者是直接例化图 15-18 中的内容。

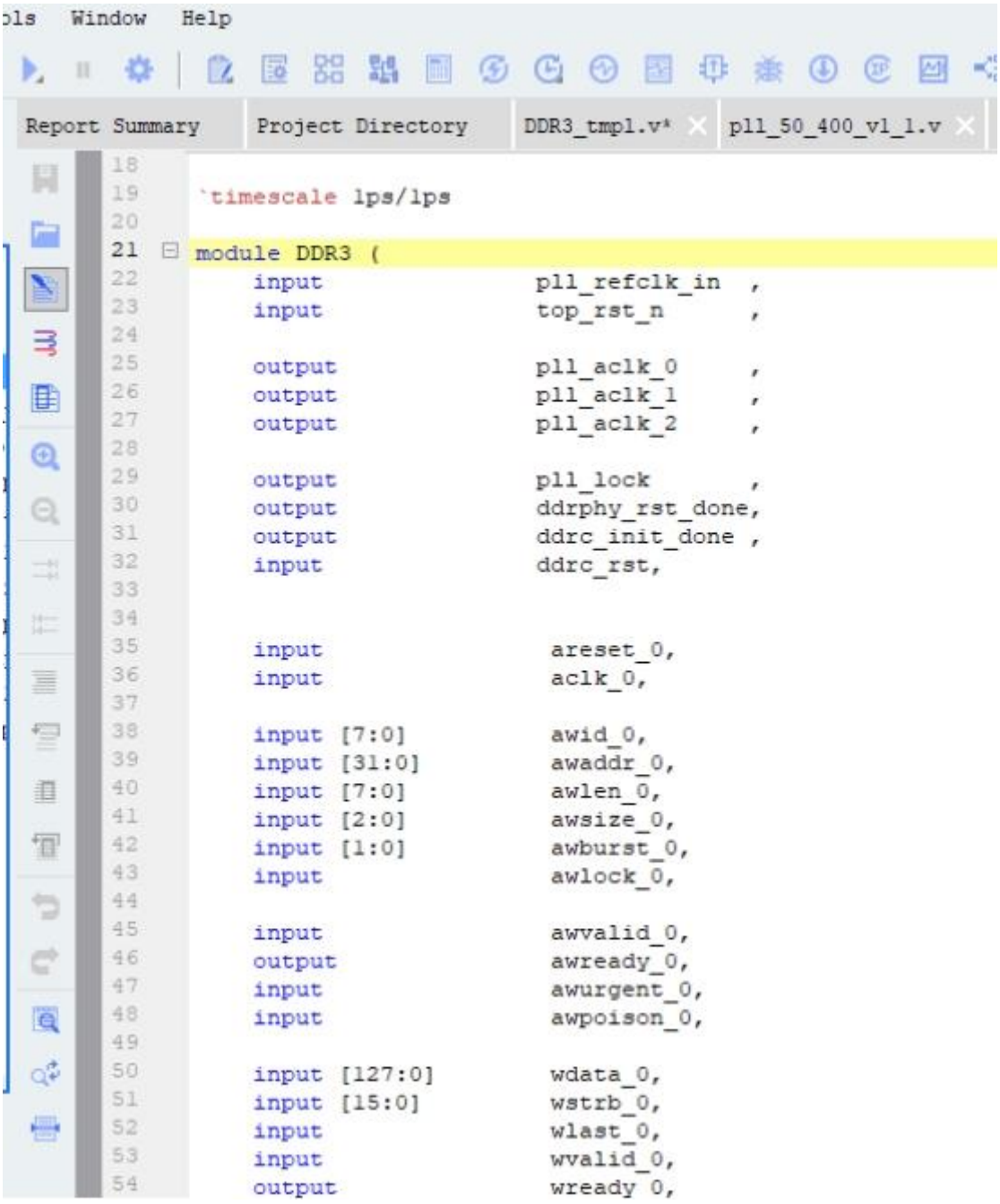


图 15-21 DDR3.V 部分内容

至此，DDR 控制器的 IP 就配置并生成完成了。

15.3DDR 控制器 AXI 接口协议简介

通过打开生成 IP 的例化模板，代码如下。

```
DDR3 the_instance_name (
  .pll_refclk_in(pll_refclk_in),        // input
  .top_rst_n(top_rst_n),                // input
```

```
    .ddrc_rst(ddrc_rst),                    // input
    .csysreq_ddrc(csysreq_ddrc),            // input
    .csysack_ddrc(csysack_ddrc),            // output
    .cactive_ddrc(cactive_ddrc),            // output
    .pll_lock(pll_lock),                    // output
    .pll_aclk_0(pll_aclk_0),                // output
    .pll_aclk_1(pll_aclk_1),                // output
    .pll_aclk_2(pll_aclk_2),                // output
    .ddrphy_rst_done(ddrphy_rst_done),      // output
    .ddrc_init_done(ddrc_init_done),        // output
    .pad_loop_in(pad_loop_in),              // input
    .pad_loop_in_h(pad_loop_in_h),          // input
    .pad_rstn_ch0(pad_rstn_ch0),            // output
    .pad_ddr_clk_w(pad_ddr_clk_w),          // output
    .pad_ddr_clkn_w(pad_ddr_clkn_w),        // output
    .pad_csn_ch0(pad_csn_ch0),              // output
    .pad_addr_ch0(pad_addr_ch0),            // output [15:0]
    .pad_dq_ch0(pad_dq_ch0),                // inout [15:0]
    .pad_dqs_ch0(pad_dqs_ch0),              // inout [1:0]
    .pad_dqsn_ch0(pad_dqsn_ch0),            // inout [1:0]
    .pad_dm_rdqs_ch0(pad_dm_rdqs_ch0),      // output [1:0]
    .pad_cke_ch0(pad_cke_ch0),              // output
    .pad_odt_ch0(pad_odt_ch0),              // output
    .pad_rasn_ch0(pad_rasn_ch0),            // output
    .pad_casn_ch0(pad_casn_ch0),            // output
    .pad_wen_ch0(pad_wen_ch0),              // output
    .pad_ba_ch0(pad_ba_ch0),                // output [2:0]
    .pad_loop_out(pad_loop_out),            // output
    .pad_loop_out_h(pad_loop_out_h),        // output
    .areset_0(areset_0),                    // input
    .aclk_0(aclk_0),                        // input
    .awid_0(awid_0),                        // input [7:0]
    .awaddr_0(awaddr_0),                    // input [31:0]
    .awlen_0(awlen_0),                      // input [7:0]
    .awsize_0(awsize_0),                    // input [2:0]
    .awburst_0(awburst_0),                  // input [1:0]
    .awlock_0(awlock_0),                    // input
    .awvalid_0(awvalid_0),                  // input
    .awready_0(awready_0),                  // output
    .awurgent_0(awurgent_0),                // input
    .awpoison_0(awpoison_0),                // input
    .wdata_0(wdata_0),                      // input [127:0]
    .wstrb_0(wstrb_0),                      // input [15:0]
    .wlast_0(wlast_0),                      // input
    .wvalid_0(wvalid_0),                    // input
    .wready_0(wready_0),                    // output
    .bid_0(bid_0),                          // output [7:0]
```

```
  .bresp_0(bresp_0),                    // output [1:0]
  .bvalid_0(bvalid_0),                  // output
  .bready_0(bready_0),                  // input
  .arid_0(arid_0),                      // input [7:0]
  .araddr_0(araddr_0),                  // input [31:0]
  .arlen_0(arlen_0),                    // input [7:0]
  .arsize_0(arsize_0),                  // input [2:0]
  .arburst_0(arburst_0),                // input [1:0]
  .arlock_0(arlock_0),                  // input
  .arvalid_0(arvalid_0),                // input
  .arready_0(arready_0),                // output
  .arpoison_0(arpoison_0),              // input
  .rid_0(rid_0),                        // output [7:0]
  .rdata_0(rdata_0),                    // output [127:0]
  .rresp_0(rresp_0),                    // output [1:0]
  .rlast_0(rlast_0),                    // output
  .rvalid_0(rvalid_0),                  // output
  .rready_0(rready_0),                  // input
  .arurgent_0(arurgent_0),              // input
  .csysreq_0(csysreq_0),                // input
  .csysack_0(csysack_0),                // output
  .cactive_0(cactive_0)                 // output
);
```

在使用 IP 前，我们先来熟悉下 IP 部分重要输入/输出端口信号。

Memory 接口信号是外部 DDR3 存储器的接口，Memory 接口信号主要包含：

表 1 Memory 接口

端口名	输入/输出	位宽	有效值	描述
pad_addr_ch0	输出	15		Memory 地址总线
pad_ba_ch0	输出	3		Bank 地址总线
pad_ddr_clk_w	输出	1		Memory 差分时钟正端
pad_ddr_clkn_w	输出	1		Memory 差分时钟负端
pad_cke_ch0	输出	1	高电平	Memory 差分时钟使能
pad_dm_rdqs_ch0	输出	2	高电平	数据 Mask
pad_odt_ch0	输出	1		On Die Termination
pad_csn_ch0	输出	1	低电平	Memory 片选
pad_rasn_ch0	输出	1	低电平	行地址 strobe
pad_casn_ch0	输出	1	低电平	列地址 strobe.
pad_wen_ch0	输出	1	低电平	写使能
pad_rstn_ch0	输出	1	低电平	Memory 复位
pad_dq_ch0	输入/输出	16		数据总线
pad_dqs_ch0	输入/输出	2		数据时钟正端
pad_dqsn_ch0	输入/输出	2		数据时钟负端
pad_loop_in	输入	1		低位温度补偿输入

pad_loop_in_h	输入	1		高位温度补偿输入
pad_loop_out	输出	1		低位温度补偿输出
pad_loop_out_h	输出	1		高位温度补偿输出

HMIC_H IP 可提供三组 AXI4 Host Port，一组 128bit，两组 64bit，由于本章实验只用到关于 AXI4 Port0 接口，则主要对 AXI4 Port0 接口的部分关键信号进行介绍。

表 2 AXI4 Port0 接口的部分关键信号

端口名	输入/输出	位宽	有效值	描述
aclk_0	输入	1		AXI Port0 输入时钟
araddr_0	输入	32		AXI Port0 读地址
arvalid_0	输入	1	高电平	AXI Port0 读地址有效信号
arready_0	输入	1	高电平	AXI Port0 读地址准备信号
rdata_0	输出	128		AXI Port0 读数据
rlast_0	输出	1		AXI Port0 读最后信号
rvalid_0	输出	1	高电平	AXI Port0 读数据有效信号
rready_0	输入	1	高电平	AXI Port0 读数据准备信号
awaddr_0	输入	32		AXI Port0 写地址
awvalid_0	输入	1	高电平	AXI Port0 写地址有效信号
awready_0	输出	1	高电平	AXI Port0 写地址准备信号
wdata_0	输入	128		AXI Port0 写数据
wlast_0	输入	1		AXI Port0 写最后信号
wvalid_0	输入	1	高电平	AXI Port0 写数据有效信号
wready_0	输出	1	高电平	AXI Port0 写数据准备信号
bresp_0	输出	2		AXI Port0 写响应
bvalid_0	输出	1	高电平	AXI Port0 写响应有效信号
bready_0	输入	1	高电平	AXI Port0 写响应准备信号

AXI 总线共 5 个独立的通道,分别是分别是 Read address channel (ARxxx)，Write address channel(AWxxx)，Read data channel(Rxxx)，Write data channel(Wxxx)，Write response channel(Bxxx)。每一个 AXI 传输通道都是单方向的。接口信号中以“aw”开头的是写地址通道信号，以“w” 开头的是写数据通道信号，以“b”开头的是写响应通道信号，以“ar”开头的是读地址通道信号，以“r”开头的是读数据响应通道信号。5 个通道分为两个事务，写事务和读事务。

其中写事务的结构图如图 15-22：

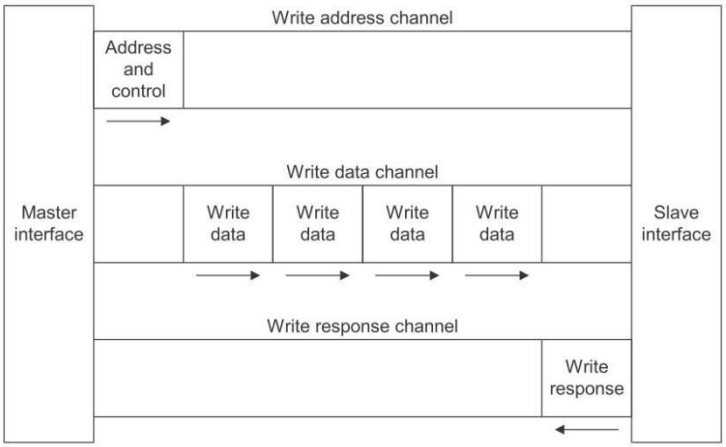


图 15-22 写事务结构图

读事务的结构图如图 15-23：

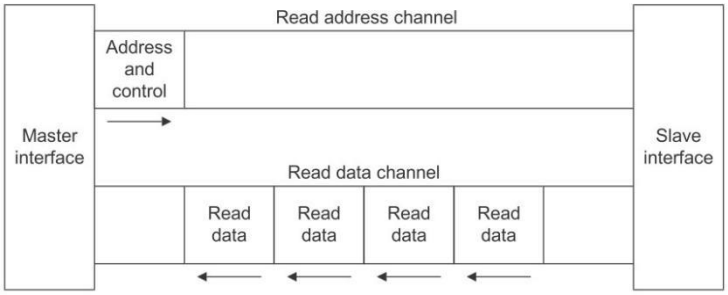


图 15-23 读事务的结构图

1. 这 5 条独立的通道都包含一个双路的 VALD、READY 握手机制。信息源通过 VALID 信号来指示通道中的数据和控制信息什么时候有效。目地源用 READY 信号来表示何时准备好接收数据。传输地址信息和数据都是在 VALID 和 READY 同时为高时有效。
2. 读数据和写数据通道都包括一个 LAST 信号，用来指明一个事物传输的最后一个数据。
3. 读/写事务都有自己的地址通道，地址通道携带着传输事务所必须的地址和信息。
4. 读数据通道传送着从设备到主机的读数据和读响应信息。读响应信息指明读事务的完成状态。
5. 写数据通路传送着主机向设备的写数据和写控制信息。写响应通道提供了设备响应写事务的一种方式。在每一次突发式写会产生一个完成信号。

下面分别对读事务和写事务的时序进行说明。

以 AXI4 Port0 为例，AXI4 接口单次写操作的时序如图 15-24 所示，当主设备发送地址和控制信息到写地址通道之后，写操作开始。然后主设备通过写数据通道发送每一个写数据，当为最后一个需要发送的数据时，主设备将 wlast_0 信号置高。当从设备接收完所有的数据时，从设备返回给主设备一个写响应标志本次写操作的结束。连续写操作与连续读操作类似，即主设备在从设备接收第一次写操作的地址后发送下一次写操作的地址。

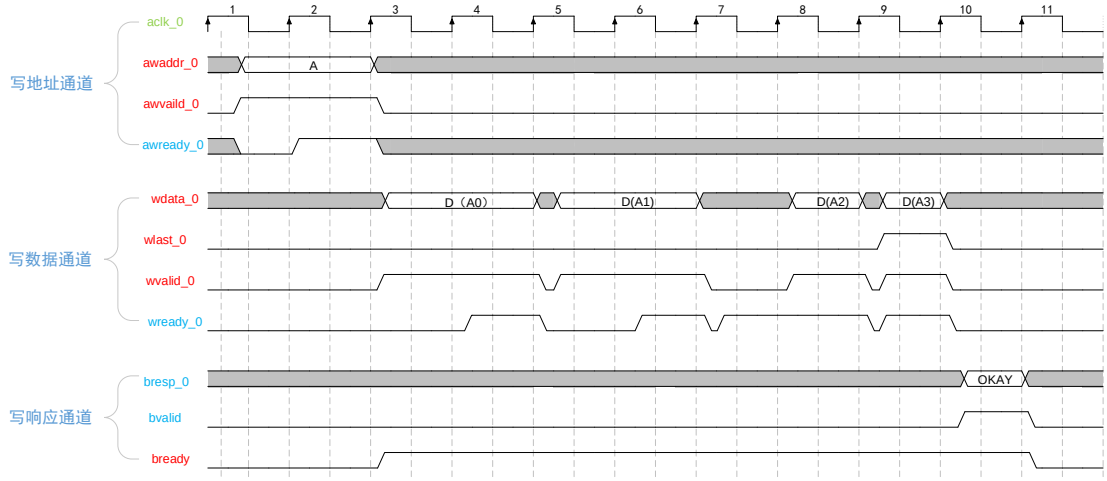


图 15-24 AXI4 单次写时序

AXI4 接口单次读操作的时序如图 15-25 所示，主设备发送地址，一个周期后从设备接收。主设备在发送地址的同时也发送了一些控制信息标志了 Burst 的程度和类型，为了保持图的清晰性，在此省略这些信号。地址总线上出现地址之后，在读数据通道上发生数据的传输。从设备一直保持 rvalid_0 信号为低，直到读数据准备好。从设备发送 rlast_0 信号标志此次读操作中最后一个数据的传输。

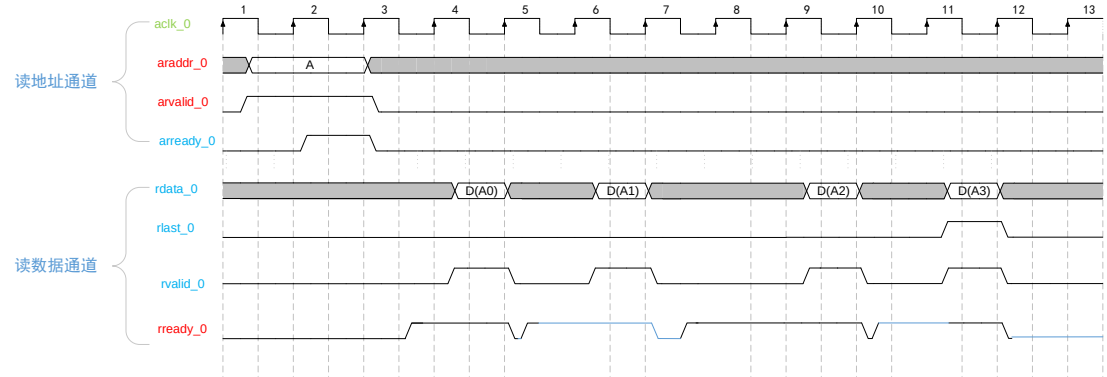


图 15-25AXI4 单次读时序

除此之外，AXI 协议支持乱序传输。每一个通过接口的事务有一个 IDtag。协议要求相同 ID tag 的事务必须有序完成，而不同 ID tag 可以乱序完成，有关乱序读写时序这里就不展开讲解。

15.4DDR 控制器 Example Design 生成

IP 核生成完成后，在 example_design 文件夹下面找到 rtl 文件夹，将 rtl 文件夹中的文件全部加载到工程的源文件下面，路径如图 15-26 所示。

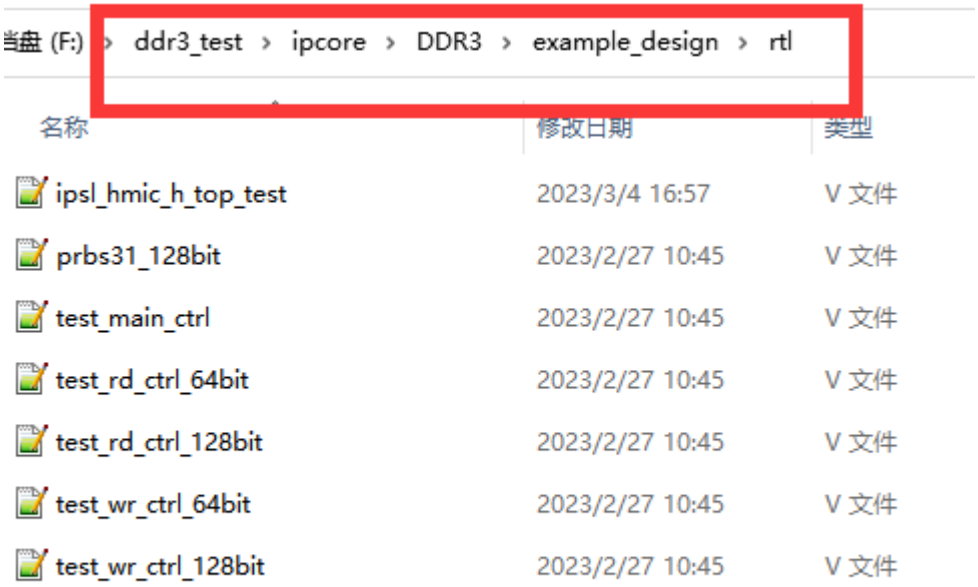


图 15-26 加载代码路径

打开 PDS 软件的 source 窗口可以看到加载的源文件以及自动生成的 ipsl_hmic_h_top_test 顶层文件，如图 15-27 所示。

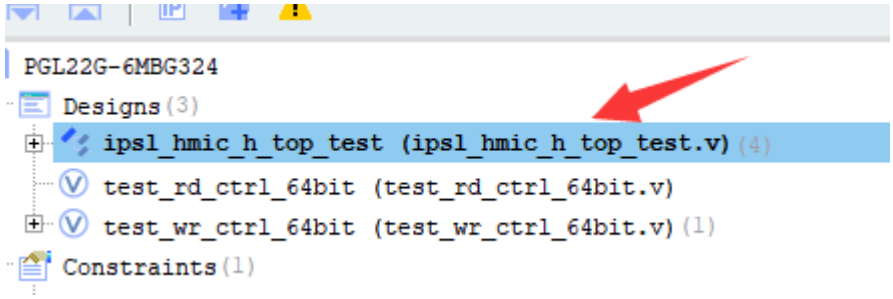


图 15-27 自动生成的设计顶层

代码设计完成后使用 PDS 软件对工程进行分析和综合，注意观察综合报告，查看是否有语法或逻辑报错，如果有报错，根据报错内容修正代码并重新进行分析和综合，直到设计分析和综合通过。

15.5 激励创建及仿真测试

15.5.1 Modelsim 手动仿真

工程编译完成后，DDR3 IP 会自动生成一个 sim 文件夹，文件夹路径如图 15-28 所示。

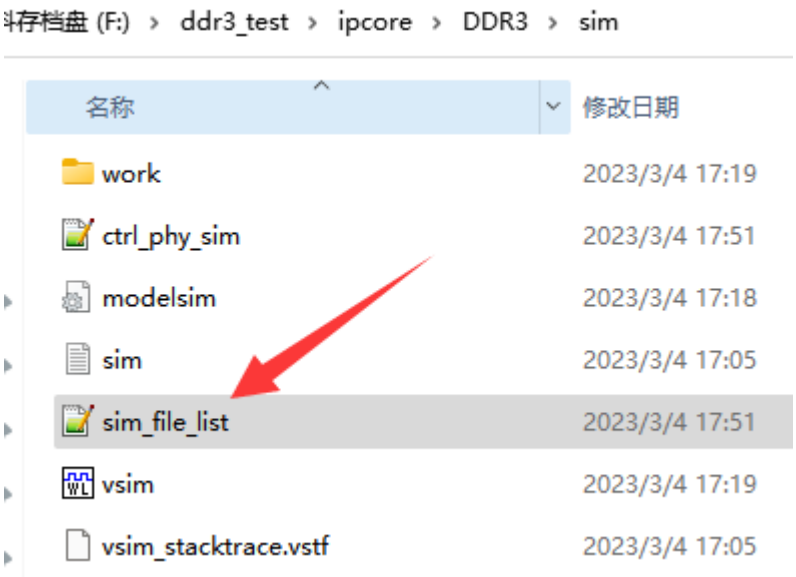


图 15-28sim 文件夹路径

用 Notepad++软件打开图 15-28 中箭头指向的 sim_file_list.f 文档可以看到图 15-29 所示内容。

```
./../example_design/rtl/test_rd_ctrl_128bit.v
./../example_design/rtl/test_wr_ctrl_128bit.v
./../example_design/rtl/test_rd_ctrl_64bit.v
./../example_design/rtl/test_wr_ctrl_64bit.v
./../example_design/rtl/ipsl_hmic_h_top_test.v
./../example_design/rtl/test_main_ctrl.v
./../example_design/rtl/prbs31_128bit.v

./../rtl/pll/pll_50_400_v1_1.v
./../rtl/ipsl_hmic_h_ddrc_apb_reset_v1_1.v
./../rtl/ipsl_hmic_h_ddrc_reset_ctrl_v1_1.v
./../rtl/ipsl_hmic_h_ddrphy_dll_update_ctrl_v1_1.v
./../rtl/ipsl_hmic_h_ddrphy_reset_ctrl_v1_1.v
./../rtl/ipsl_hmic_h_ddrphy_training_ctrl_v1_1.v
./../rtl/ipsl_hmic_h_ddrphy_update_ctrl_v1_1.v
./../rtl/ipsl_hmic_h_ddrc_top_v1_1.v
./../rtl/ipsl_hmic_h_phy_top_v1_1.v
./../rtl/ipsl_hmic_h_phy_io_v1_1.v
./../DDR3.v
```

图 15-29 代码内容

这里的文件位置是 sim 文件的相对路径，“./”表示为 sim 文件的当前位置，“../”表示为 sim 文件的上一级。举个例子“./../example_design/rtl/prbs31_128bit.v ”表示为 sim 文件所在当前路径上级 example_design 的文件夹下的 prbs31_128bit.v 文件。

DDR3 IP 设置完成后，在该工程的文件夹下面还自动生成了顶层仿真文件，路径图 15-30 所示。

图 15-30 仿真文件路径

如果不需要设置仿真结束时间，可以将 ddr_test_top_tb.v 文件中代码注释或者删除掉，如图 15-31 所示。

```
initial begin

//reset the bu_top
#10000 bu_top_rst_n = 1'b0;
#50000 bu_top_rst_n = 1'b1;
//$display("%t keyboard reset sequence finished!", $time);

//@ (posedge ddr_init_done);
//$display("%t ddr_init_done is high now!", $time);
//#100000000;
//$finish;
end
```

图 15-31 注释代码

为避免仿真时，Modelsim 软件会出现闪退现象，可以找到 ctrl_phy_sim.tcl 文件，文件所在路径如图 15-32 所示。

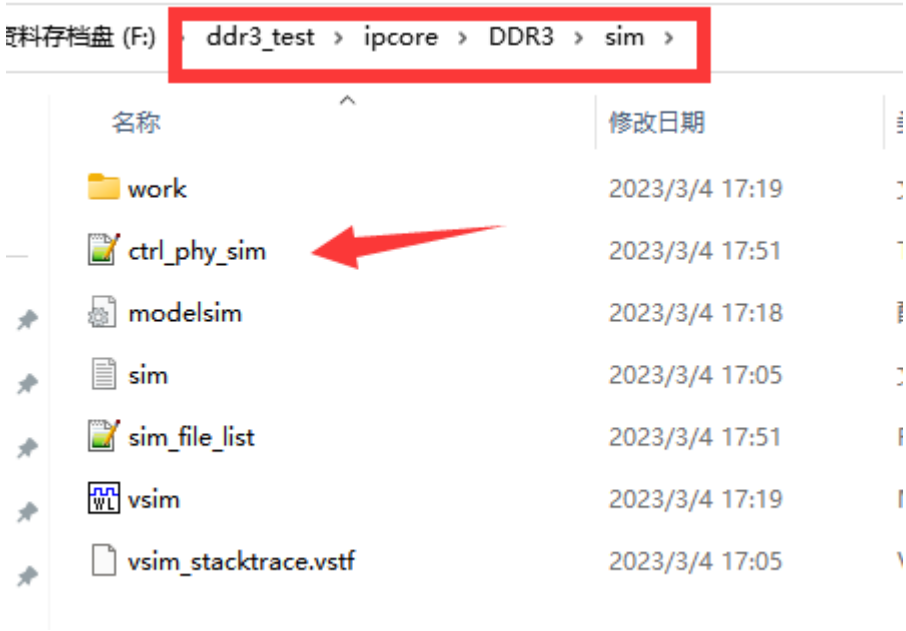


图 15-32 ctrl_phy_sim.tcl 文件所在位置

用 Notepad++软件打开，将其中“vsim -suppress 3486,3680,3781 +nowarn1 -c -sva -lib work ddr_test_top_tb -l sim.log”注释或删除掉，改成“vsim -voptargs="+acc" -suppress 3486,3680,3781 -c ddr_test_top_tb -L work -l sim.log ”具体修改如图 15-33 红框中所示。

```
quit -sim

if {[file exists work]} {
  file delete -force work
}

  vlib  work
  vmap  work work

set LIB_DIR  D:/pango/PDS_2021.4-SP1.2/ip/system_ip/ipsl_hmic_h/ipsl_hmic_h_eval/ipsl_hmic_h/../

vlib work
vlog -sv -work work -mfcu -incr -f ../sim/sim_file_list.f -y $LIB_DIR +libext+.v +incdir+../exam
vsim -voptargs="+acc" -suppress 3486,3680,3781 -c ddr_test_top_tb -L work -l sim.log
```

图 15-33 修改代码

双击打开 Modelsim 软件，点击 File 选项，选择 Change Directory，如图 15-34 所示。

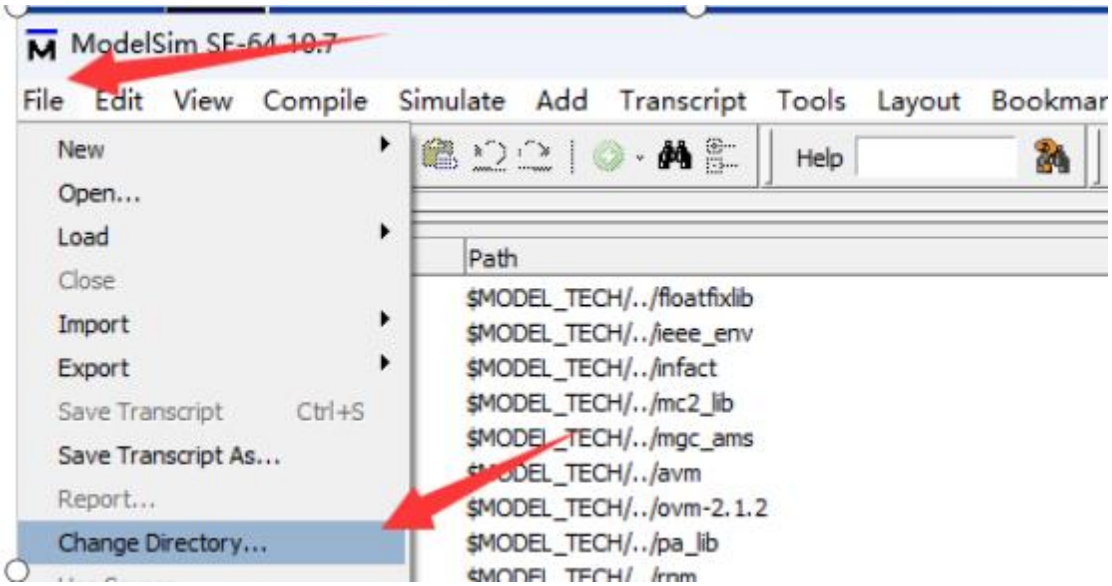


图 15-34 设置仿真路径

将路径设置为图 15-35 中箭头指向的文件夹。

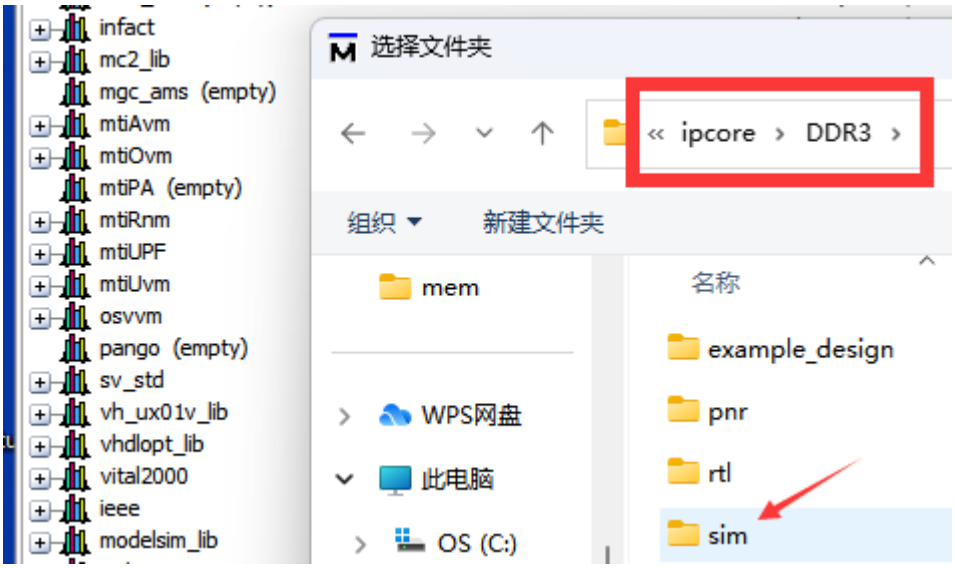


图 15-35 仿真路径

点击选择按钮，Modelsim 软件会自动跳到信息框中，信息框内容如图 15-36 Modelsim 软件信息框所示。

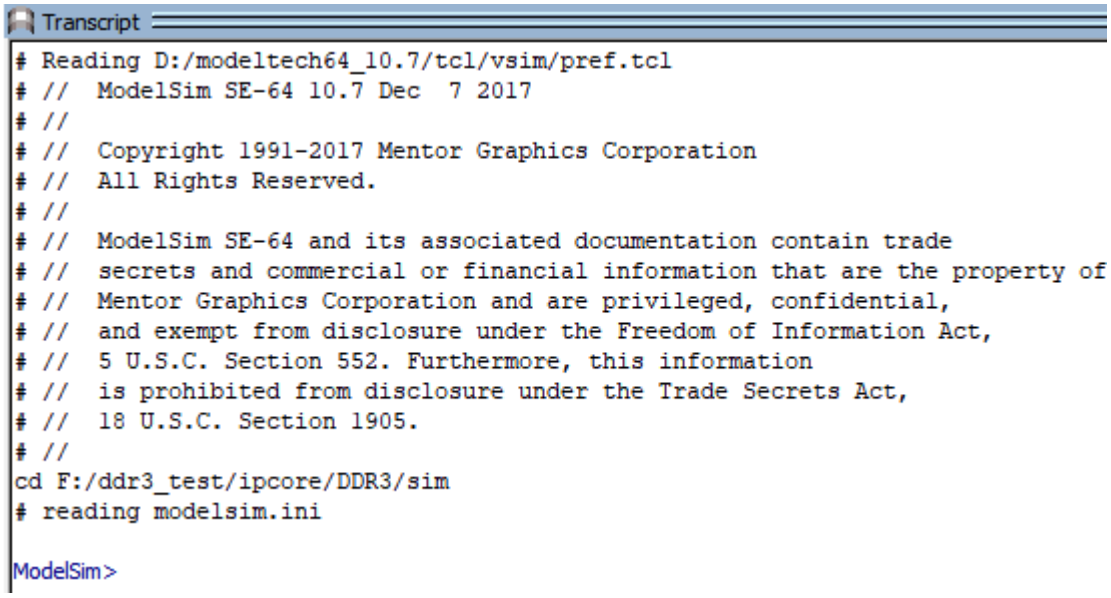


图 15-36 信息框

在信息框中输入指令 ls 如图 15-37 所示，然后点击回车会看到信息框中弹出多个信息。

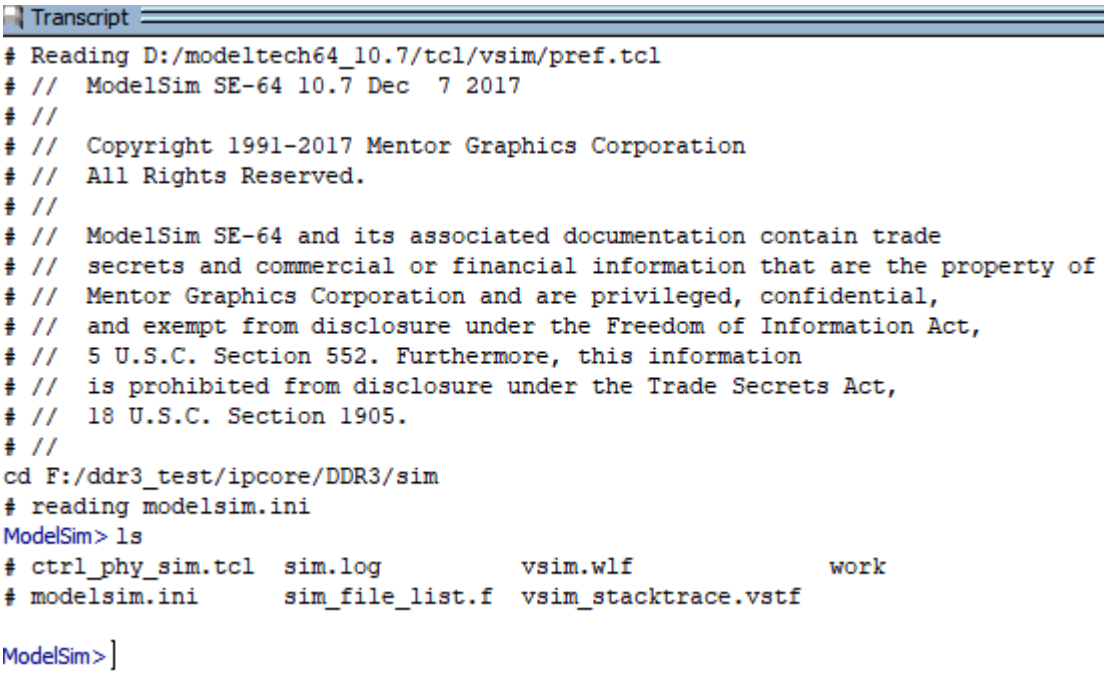


图 15-37 输入 ls 信息框

此时我们再输入 do ctrl_phy_sim.tcl 指令，然后点击回车，如图 15-38 所示。

```
ModelSim> do ctrl_phy_sim.tcl
# Model Technology ModelSim SE-[illegible] 10.7 Lib Mapping Utility 2017.12 Dec  7 2017
# vmap work work
# Modifying modelsim.ini
# D:/pango/PDS_2021.4-SP1.2/ip/system_ip/ipsl_hmic_h/ipsl_hmic_h_eval/ipsl_hmic_h/../../../../../arch/vendor/pang
# ** Warning: (vlib-34) Library already exists at "work".
# Model Technology ModelSim SE-64 vlog 10.7 Compiler 2017.12 Dec  7 2017
# Start time: 10:32:05 on Mar 01,2023
# vlog -reportprogress 300 -sv -work work -mfcu -incr -f ../sim/sim_file_list.f -y D:/pango/PDS_2021.4-SP1.2/ip/s
# -- Compiling module ddr_test_top_tb
```

图 15-38 输入 do ctrl_phy_sim.tcl 指令框

大约等待五六秒左右的时间，最终页面将跳转到 sim 窗口。

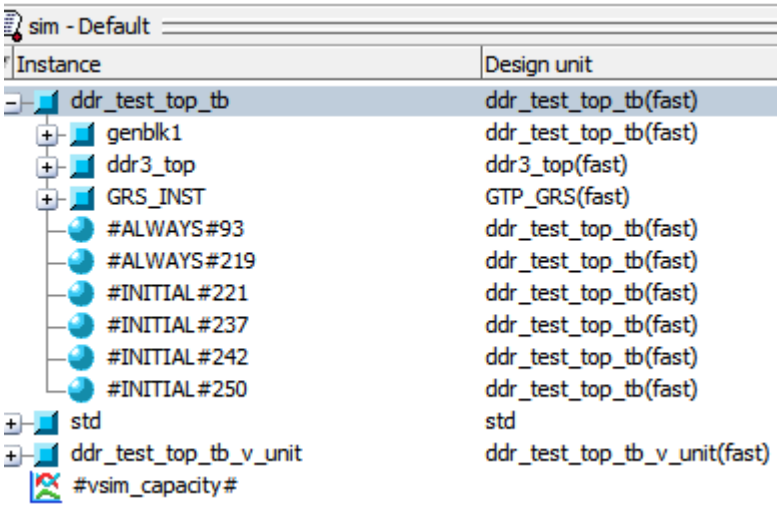


图 15-39 sim 窗口

将想要观察的波形添加到仿真页面，这里我们选择观察 DDR3 IP 模块中 ddrc_init_done 信号是否被置高，及 DDR3 存储器初始化和校准是否完成。观察 DDR3 里的写入/读出数据，通过比较写入和读出的数据是否相等验证其功能是否正常。

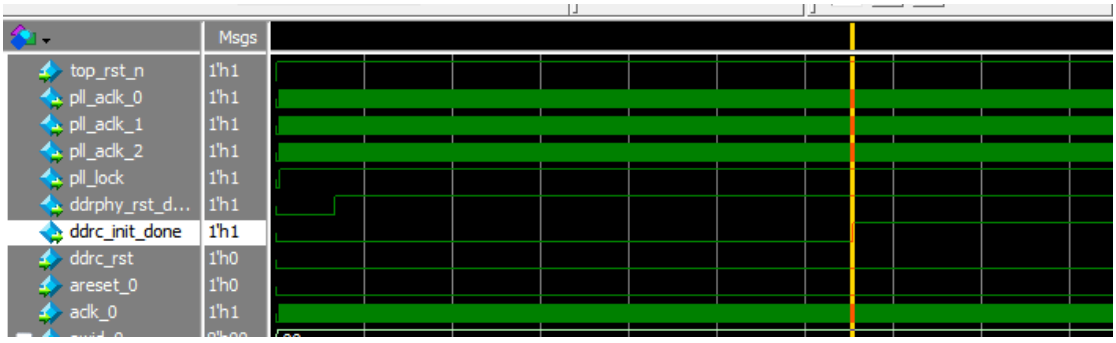


图 15-40 ddrc_init_done 信号波形

在仿真大概 70us 后，ddrc_init_done 信号被置高，说明这个时候对 DDR3 存储器初始化和校准已经完成。

仿真结束后，随便找一处写操作过程 AXI 总信号的变化情况。可以看出

axi_awsize 为 4，每个数据的字节数为 16（等于 2^axi_awsize=2^4 字节），即数据位宽为 128bit，这个与我们 IP 配置 AXI 接口的数据位宽是一致的。

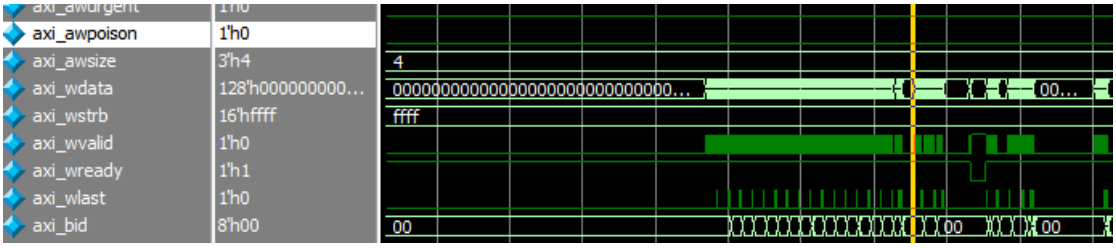


图 15-41 仿真位宽和 AXI 接口配置

观察本次写操作的的写数据通道的波形如图 15-42 所示。

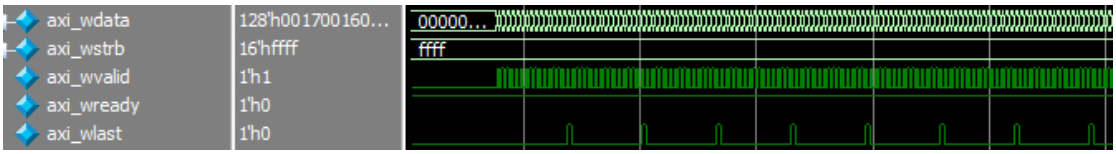


图 15-42 写操作的写数据通道波形

写数据通道写入数据之后，axi_wvalid 和 axi_wready 同时为高表示写数据有效，当写最后一个数据的时候 axi_wlast、axi_wvalid 和 axi_wready 是同时为高的。axi_wstrb 等于 16'hffff，表示写入的 16 个字节全部有效。

类似的方式，对读操作过程中 AXI 信号波形进行观察。首先找一次读操作过程波形如图 15-43 所示

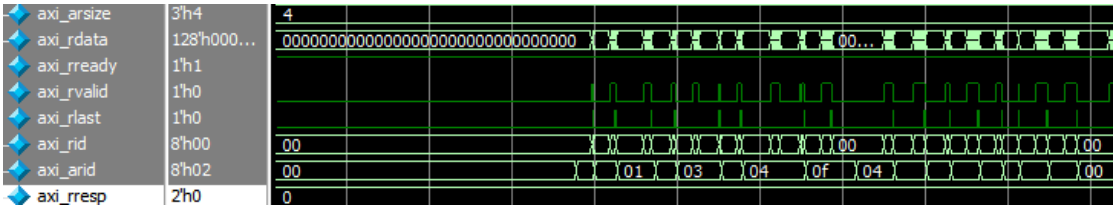


图 15-43 读地址通道进行分析波形

每个数据的字节数为 16（等于 2^axi_arsize = 2^4 字节），即数据位宽为 128bit，这个与我们 IP 配置 AXI 接口的数据位宽是一致的。

观察本次突发写操作响应通道的波形如图 15-44 所示。

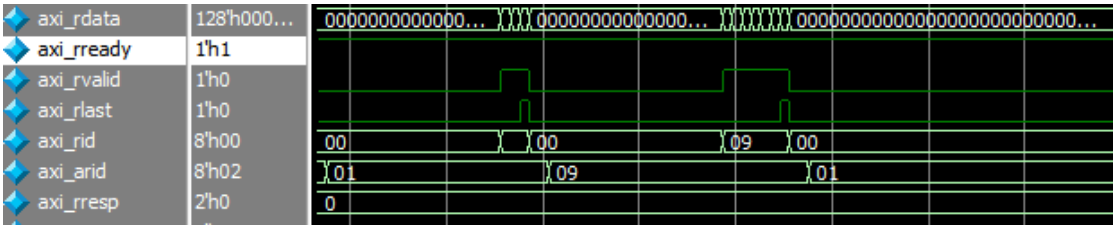


图 15-44 读操作响应通道波形

读数据响应通道读出数据之后，axi_rvalid 和 axi_rready 同时为高表示读出数据有效，当读最后一个数据的时候 axi_rlast、axi_rvalid 和 axi_rready 是同时为高的，axi_rresp 等于 0 表示本次读操作的响应状态是 OK 的。

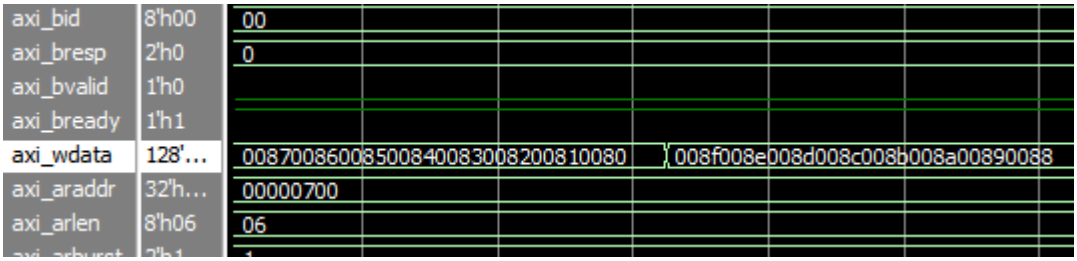


图 15-45 写数据放大波形

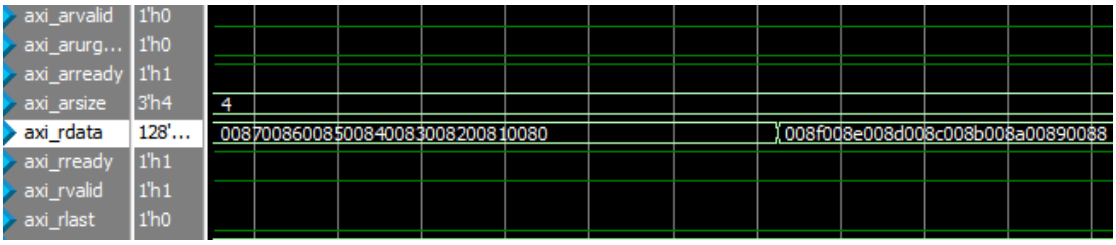


图 15-46 读数据放大波形

对比图 15-45 和图 15-46 可以看出写数据以及读数据一致，可以看出 DDR3 IP 是出于正常工作状态。

可以通过多个读/写操作，观察 AXI 总线信号的变化情况，对 AXI 协议能有更深的理解。

15.5.2PDS 联合 Modelsim 自动仿真

工程编译完成之后 DDR3 IP 会自动生成一个 sim 文件夹如图 15-35 所示，在用 Notepad++软件打开 sim_file_list 文档，在文档的最后找到图 15-47 所示内容。

```
206    D:/pango/PDS_2021.4
207    D:/pango/PDS_2021.4
208    D:/pango/PDS_2021.4
209    D:/pango/PDS_2021.4
210    D:/pango/PDS_2021.4
211    D:/pango/PDS_2021.4
212
213    +define+SIMULATION
214    +define+den4096Mb
215    +define+sg25E
216    +define+x16
217
```

图 15-47 sim_file_list 文档部分内容

在 PDS 软件的上方工具栏中找到 Project 按钮，点击 Project 按钮，再点击

Project Setting 选项，如图 15-48 所示。

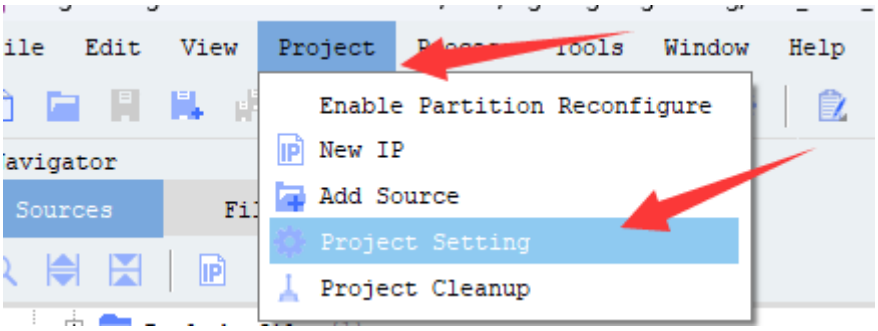


图 15-48 Project Setting 选项

点击 Project Setting 按钮之后会跳转到 Project Setting 页面如图 15-49 所示，再点击页面中 Simulation 按钮。

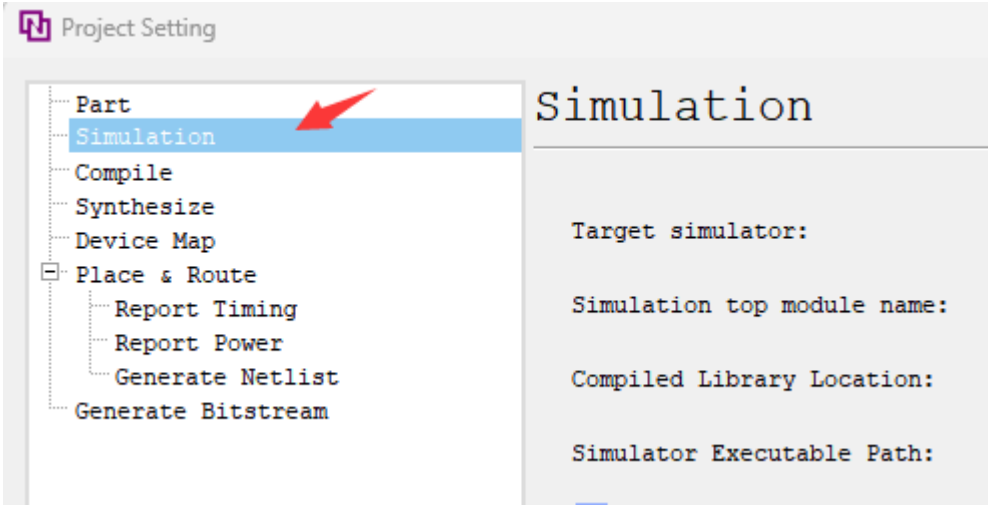


图 15-49 Project Setting 页面

点击图 15-49 中 Simulation 按钮之后，再点击图 15-50 中箭头指向位置,将“Verilog options”选项的路径定位到方框中的位置，顺便将该文件夹（mem 文件夹内容替换成本章工程中 mem 文件夹中内容）。

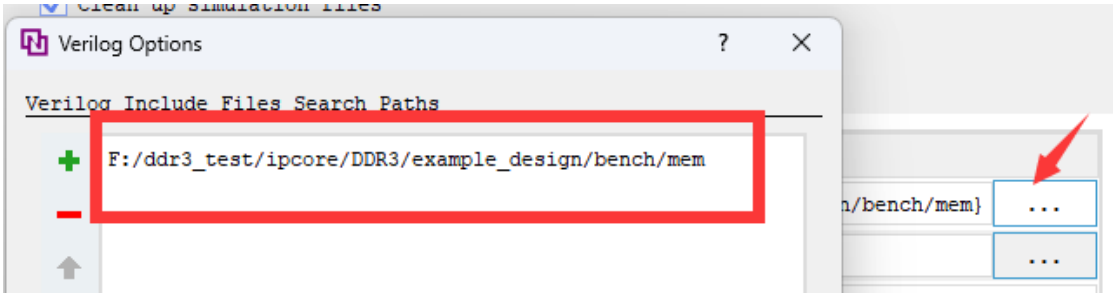


图 15-50 Compilation 页面

点击图 15-51 中箭头指向的“+”将图 15-47 中内容依次复制到图 15-51 页面中，

然后点击图 15-51 中的“OK”按钮。

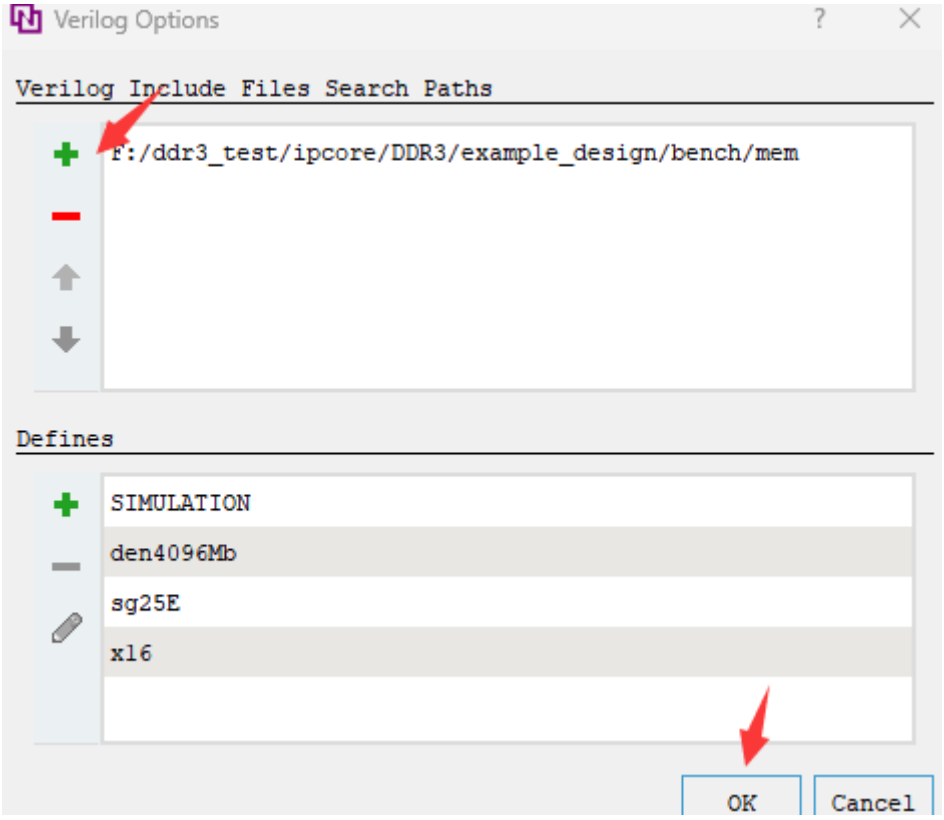


图 15-51 Generics/Parameters options 页面

PDS 软件会返回到图 15-52 中，在箭头所指向的“Modelsim.compile.vcom.more_options”选项栏后面填写-sv（表示仿真软件用 system verilog 语言进行仿真）。

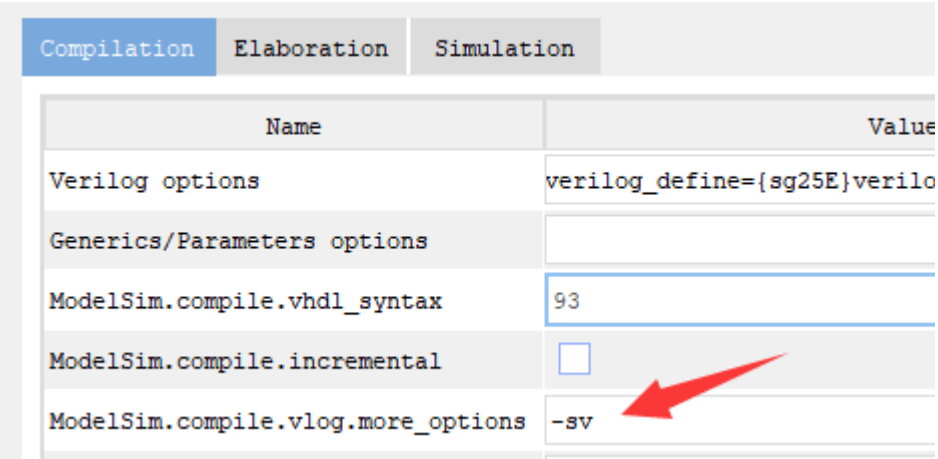


图 15-52 Compilation 页面

再点击图 15-52 页面中的“Elaboration”选项按钮，进入“Elaboration”页面。点击 ModelSim.elsaboration.acc”选项栏后面箭头指向的框，点击之后框内

会出现对号，最后点击 OK，完成设置。

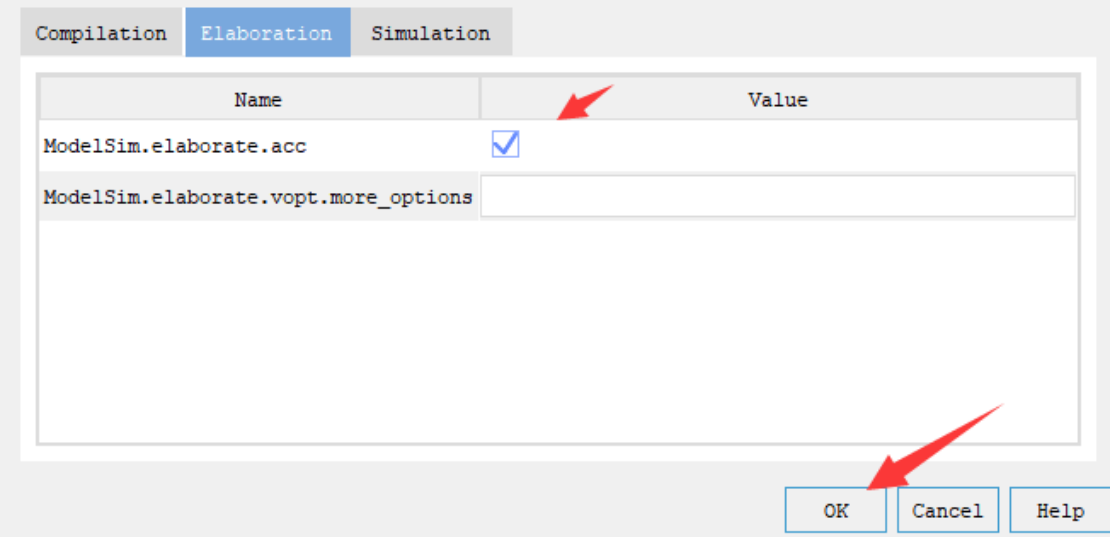


图 15-53 “Elaboration”页面

将图 15-30 中的测试文件以及涉及到的相关源文件都加载到仿真工具栏下如图 15-54 所示。

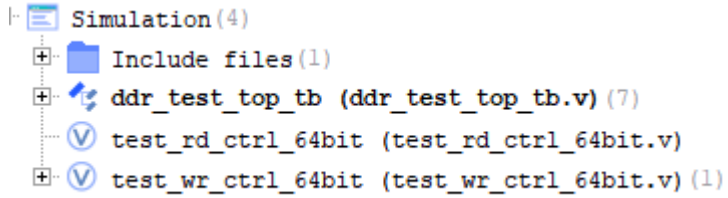


图 15-54 仿真页面

最后点击图 15-55 箭头指向的仿真按钮，进行工程文件的仿真。

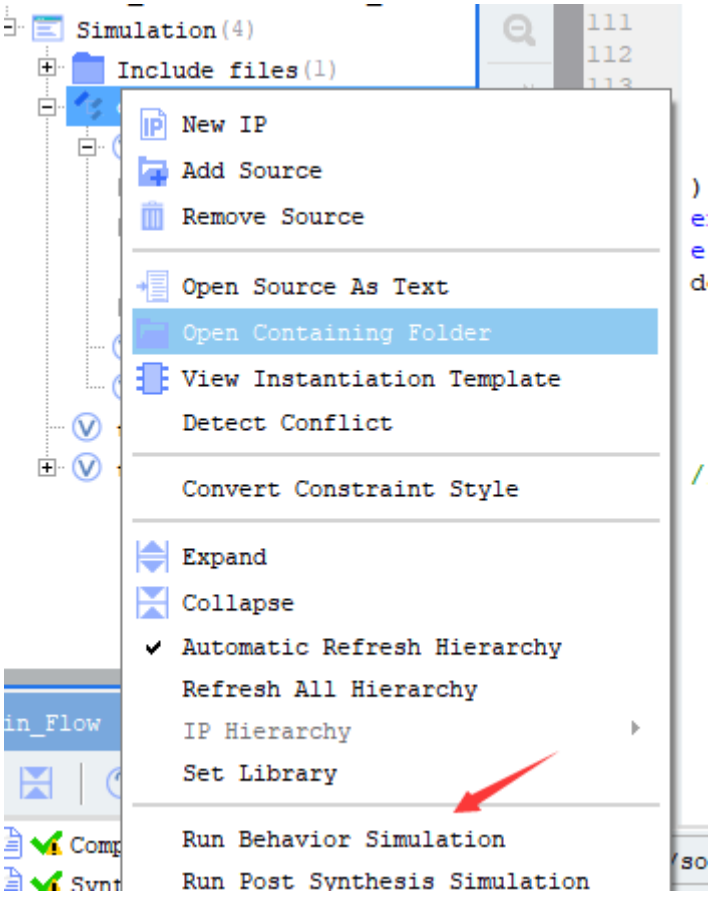


图 15-55 仿真按钮

最后对于如何分析仿真波形，这里我们不再赘述。

15.6板级调试与验证

实验的板级验证环节，主要验证以下几个目标：

1. 能否正确将生成的 bit 文件下载到 PGL22G 开发板；
2. 下载完成后能否正确显示 PGL22G 开发板 LED0 的点亮实验现象；

系统所需硬件：

1. PGL22G 开发板；
2. 电源电缆一根；
3. 硬件条件符合实验要求，具有完全开发功能的 PC 机一台；

15.6.1添加 I/O 约束

通常，一个设计中的 FPGA 不会是独立使用的，FPGA 一定会与其他外设、

接口相连接，比如时钟，按键等。因此，FPGA 设计需要指定对应的 IO 引脚位置信息。

将 I/O 约束写入到约束文件中，由于约束引脚过多，这里我们不做展示，可以直接将我们工程中 DDR3 IP 相关引脚的约束代码约束进自己的工程的约束文件中，我们的约束文件所在路径如图 15-56 所示。

图 15-56 约束文件位置

除了将 ddr3 相关的约束代码复制进来外，还要再将如下约束复制进我们工程的约束文件中，约束语句如下。

```
define_attribute{i:ddr3_ctrl_2port/DDR3/u_pll_50_400/u_pll_e1}{PAP_LOC} {PLL_82_71}
```

上述约束语句格式是固定的，其中只有第一个{}和第三个{}内的代码是用户修改的。第一个{}内“i：”后是的代码是用户定义的，此处代码是为了定位到所需约束的 PLL 在工程中的位置。第三个{}内为约束 PLL 编号，PLL 编号可从 tcl 中查找。

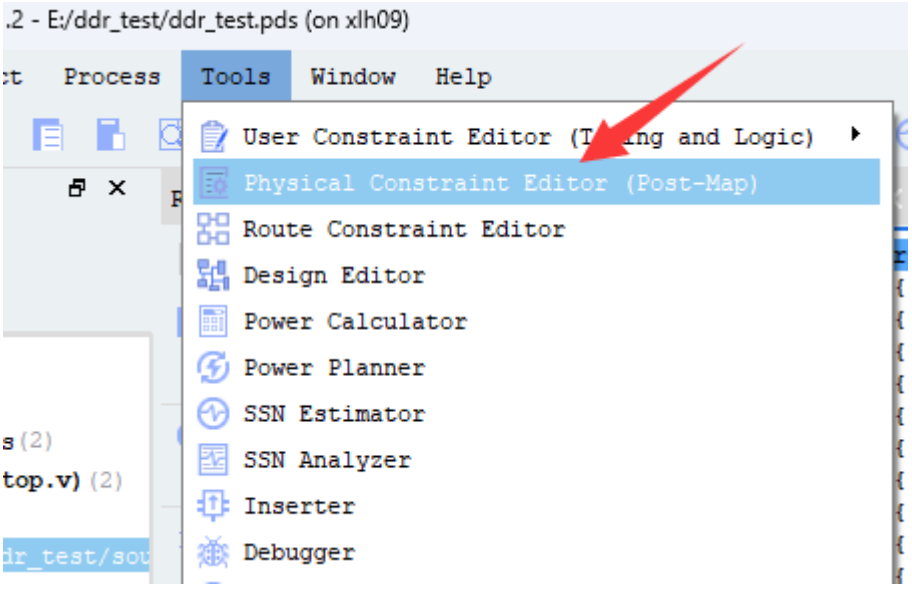


图 15-57 “Physical Constraint Editor ”页面

点击“Physical Constraint Editor”按钮之后可以看到 TCL 页面。再点击图 15-58 中箭头指向的“TCL”按钮。在图 15-58 中六个方框分别对应六个 PLL 编

号(由上到下分别表示 PLL0~PLL5)，点击我们需要约束的 PLL，即可在下方看到其对应编号。

图 15-58TCL 页面

由于我们所用的芯片是 PLG22G 系列，所以我们将 PLL 约束至 PLL4 编号即可。点击图 15-59 页面中的绿色方框，可以查看 PLL 对应编号。

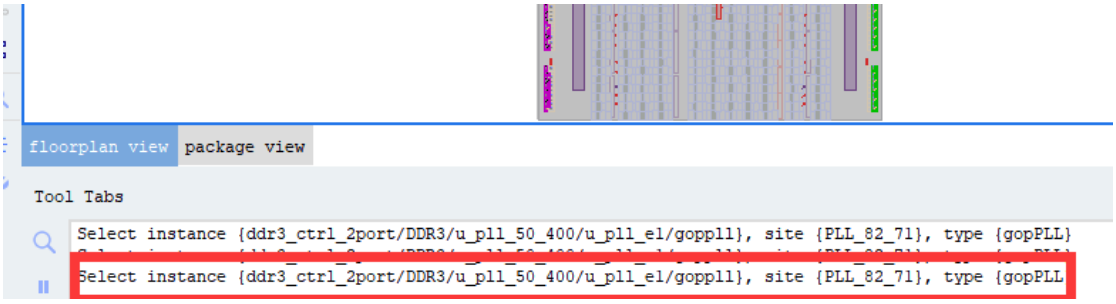


图 15-59 PLL 对应编号

15.6.2下载与验证

将下载器一端连接电脑，另一端与开发板上的 JTAG 下载口连接，连接电源线，硬件连接如图 15-60 所示。

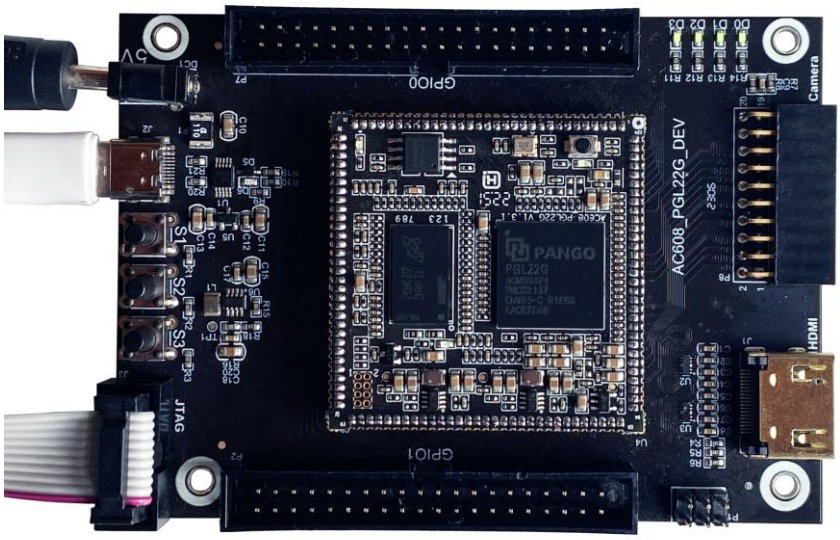


图 15-60 硬件连接图

打开开发板电源开关，并下载该工程对应的比特流文件。实验现象如图 15-61 所示。

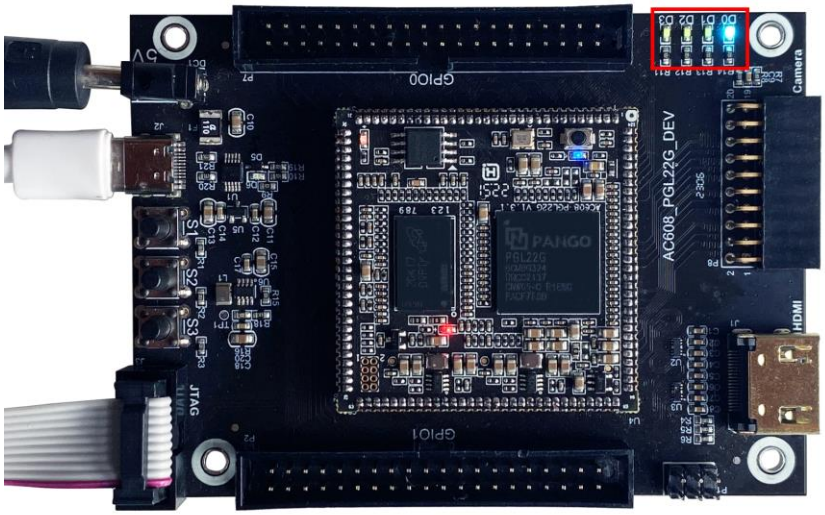


图 15-61 实验现象

将比特流文件下载到开发板后可以观察到对应的现象。LED1 ，LED2 LED3 处于熄灭状态， LED0 处于亮的状态（这是因为我们开发板设计为低电平点亮）。LED0 处于常亮状态说明读数据正确，LED1 熄灭表明硬件复位完成，LED2 熄灭表明 PLL 稳定，这说明开发板 DDR3 相关硬件是能正常工作的。

LED3 处于熄灭状态表明我们的 DDR3 IP 初始化成功。这些现象均表明 DDR3 IP 是正常工作的。

15.7常见问题

1. DDR3 IP 相关引脚的驱动电平为 1.5V，若电平驱动设置为 3.3V 会导致 DDR3 IP 不能正常工作
2. PLL 的相关约束也要添加到约束文件中，否则会导致工程不能运行。

15.8总结

本章介绍了 DDR3 IP 的组成及各部分的功能，使用 IP 核实现一个点灯实验，从而判断 DDR3 IP 能否正常工作。同时，使读者对本开发板有一个上手体验。建议读者能够跟随本实验内容，完整的进行整个实验。